Operating System/2™ Programming Tools and Information Version 1.2

I/O Subsystems and Device Support
Volume 2
Presentation Driver Interfaces

64F0283

Operating System/2™
Programming Tools
and Information
Version 1.2

I/O Subsystems and Device Support
Volume 2
Presentation Driver Interfaces

First Edition (September 1989)

The following paragraph does not apply to the United Kingdom or any country where such provisions are inconsistent with local law:

INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This publication could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time.

It is possible that this publication may contain reference to, or information about, IBM products (machines and programs), programming, or services that are not announced in your country. Such references or information must not be construed to mean that IBM intends to announce such IBM products, programming, or services in your country.

Any reference to an IBM licensed program or other IBM product in this publication is not intended to state or imply that only IBM's program or other product may be used.

Requests for technical information about IBM products should be made to your IBM Authorized Dealer or your IBM Marketing Representative.

© Copyright International Business Machines Corporation 1988, 1989. All rights reserved.

Note to US Government users — Documentation related to Restricted Rights — Use, duplication, or disclosure is subject to restrictions set forth in GSA ADP Schedule Contract with IBM Corp.

Special Notices

The following names, used in this publication, are trademarks or registered trademarks of International Business Machines Corporation in the United States and/or other countries:

IBM
C/2
Macro Assembler/2
Operating System/2
OS/2
Presentation Manager
Systems Application Architecture.

The following names, used in this publication, are trademarks or registered trademarks of other companies:

CodeView	Microsoft Corporation
PostScript	Adobe Systems Incorporated.

IBM may have patents or pending patent applications covering subject matter in this document. The furnishing of this document does not give you any license to these patents. You can send license enquiries, in writing, to the IBM Director of Commercial Relations, IBM Corporation, Armonk, NY 10504.

Preface

This book contains information about presentation drivers for IBM Operating System/2. It describes the internal interfaces between the Presentation Manager and the driver and between the driver and I/O Subsystems such as the Spooler.

Detailed descriptions of control structures, data structures, and I/O formats have been included where they are needed to understand and use the interfaces. For further information about structures and defined values, see the *C/2 Bindings Reference* for Presentation Manager and the header files (such as PMWIN.H and PMDDI.H) that are supplied with the OS/2 1.2 Programming Tools.

This book is intended for the system programmer. The reader needs an understanding of operating systems and 80286 architecture. For information about the concepts and implementation of OS/2 programming, see the relevant publication in the OS/2 library (see page vi).

In this book:

Chapter 10. Presentation Drivers Overview
This chapter shows where presentation drivers reside in the system, what they do, and gives some general guidance.

Chapter 11. Exported Entry Points
This chapter defines the entry points that a presentation driver must export to the system so that an application program can query the driver and the system can enable it.

Chapter 12. Mandatory Functions for All Devices
This chapter describes the internal functions that must be supported by handling routines in the presentation driver. (The functions described in this chapter must be supported by all presentation drivers.)

Chapter 13. Mandatory Functions for Display Devices
This chapter describes additional internal functions that must be supported by the presentation driver when the driver is for a display device.

Chapter 14. Simulated Functions
This chapter describes internal functions that are supported in the system's graphics engine and can be hooked by the presentation driver. (Drivers might need to hook these functions to exploit special features in the device.)

Chapter 15. Graphics Engine Internal Functions
This chapter describes internal functions that are supported in the system's graphics engine and can be called by the presentation driver.

Chapter 16. System Functions
This chapter describes system functions that are not part of the OS/2 API but can be called by presentation drivers.

Chapter 17. Presentation Manager Spooler
This chapter describes spooler (including Queue Processor and Print Manager) functions that a presentation driver calls to manage spooled files for output.

Chapter 18. File System Emulation
This chapter describes DOS-emulation functions that a presentation manager calls to send data, through the kernel device driver, to the physical device.

Appendix A. Syntax Conventions
This appendix shows the conventions that have been used for the parameter names used in the Presentation Manager library.

Appendix B. Initialization File
This appendix describes entries in the OS2SYS.INI file that are used by presentation drivers but, because they serve no purpose at the API, are not described in the other OS/2 books.

A glossary and an index are included at the back of this book.

The OS/2 1.2 Library and Related Products

OS/2 Product

IBM Operating System/2
Standard Edition
Version 1.2

Getting Started
Using Advanced Features
Product Information

6024926 3.5″ diskette
6024930 5.25″ diskette

OS/2 Programming Tools and Information

IBM Operating System/2
Version 1.2

Installation
Programming Overview
Programming Guide
Building Programs
Dialog Tag Language Guide and Reference
Dialog Manager Guide and Reference
Dialog Manager and Dialog Tag Language Reference Summary
Programming Reference (3 books)
Bindings Reference for Presentation Manager (4 books for COBOL/2, FORTRAN/2, C/2, and Macro Assembler/2)
I/O Subsystems and Device Support (2 books)
Systems Application Architecture Common User Access: Advanced Interface Design Guide

6024929

Separate Order (no charge)

Keyboards and Code Pages

6280345

Available Separately

Command Reference

6024928

Service Coordinator's Guide

15F2214

Programming Languages

IBM Basic Compiler/2
Version 1.0 6280179

IBM Macro Assembler/2
Version 1.0 6280181

IBM Pascal Compiler/2
Version 1.0 6280183

IBM FORTRAN/2
Version 1.02 6280185

IBM COBOL/2
Version 1.0 6280207

IBM C/2
Version 1.1 6280284

Systems Application Architecture

An Overview

GC26-4341

Writing Applications: A Design Guide

SC26-4362

Common User Access: Advanced Interface Design Guide
SC26-4582

Common User Access: Basic Interface Design Guide
SC26-4583

Common Programming Interface

C Reference - Level 2

SC09-1308

COBOL Reference

SC26-4354

Dialog Reference

SC26-4356

FORTRAN Reference

SC26-4357

Procedures Language Reference
SC26-4358

Presentation Reference

SC26-4359

Contents

Chapter 10. Presentation Drivers Overview 10-1
Display Devices 10-1
Printers and Plotters 10-1
Functions 10-3
Presentation Driver Interface 10-3
Dynamic Link Library Functions 10-3
Device Context 10-4
Device Types 10-5
Data Types 10-5
Instance Data 10-5
Program Stack 10-5
Calling Conventions 10-5
Function Number and Command Flags 10-6
Design Considerations 10-7
Sample Presentation Driver 10-7
General Considerations 10-7
Interrupts 10-8
Drawing to Display Devices 10-8
Exit List Processing 10-8
Allocating Memory 10-9
Protecting Objects 10-9
Protecting the DC 10-9
Bit Map Simulation 10-10
Banding 10-11
Error Strategy 10-12

Chapter 11. Exported Entry Points 11-1
OS2_PM_DRV_DEVMODE 11-2
OS2_PM_DRV_DEVICENAMES 11-4
OS2_PM_DRV_ENABLE 11-5
Enable subfunction 01H – FillLogicalDeviceBlock 11-7
Enable subfunction 02H – FillPhysicalDeviceBlock 11-9
Enable subfunction 04H – DisablePhysicalDeviceBlock 11-11
Enable subfunction 05H – EnableDeviceContext 11-12
Enable subfunction 06H – DisableDeviceContext 11-14
Enable subfunction 07H – SaveDCState 11-15
Enable subfunction 08H – RestoreDCState 11-16
Enable subfunction 09H – ResetDCState 11-17
Enable subfunction 0AH – CompleteOpenDC 11-18
Enable subfunction 0BH – BeginCloseDC 11-19
MoveCursor 11-20
Programming Considerations 11-20

Chapter 12. Mandatory Functions for All Devices 12-1
Line Functions 12-2
GreGetCurrentPosition 12-5
GreSetCurrentPosition 12-6
GrePolyLine 12-8
GreDrawLinesInPath 12-10
GrePolyShortLine 12-12
GrePolyScanLine 12-14
Text Functions 12-16
GreCharString 12-17
GreCharStringPos 12-19
GreQueryTextBox 12-22
GreQueryCharPositions 12-24
GreQueryWidthTable 12-26
Marker Functions 12-27
GrePolyMarker 12-28

Bit Map Functions 12-30
Bit-Map File Formats 12-30
GreDeviceCreateBitmap 12-31
GreDeviceDeleteBitmap 12-33
GreDeviceSelectBitmap 12-34
GreBitblt 12-35
GreGetPel 12-40
GreSetPel 12-41
GreImageData 12-42
GreDrawBorder 12-43
GreGetBitmapBits 12-46
GreSetBitmapBits 12-48
Attribute Functions 12-50
Attribute and Bundle Definitions 12-50
GreDeviceGetAttributes 12-63
GreGetPairKerningTable 12-64
GreDeviceSetAttributes 12-65
GreDeviceSetGlobalAttribute 12-67
Device Functions 2 12-68
GreNotifyClipChange 12-69
GreNotifyTransformChange 12-70
GreRealizeFont 12-72
GreErasePS 12-75
GreGetStyleRatio 12-77
GreSetStyleRatio 12-78
GreDeviceQueryFontAttributes 12-79
GreDeviceQueryFonts 12-80
Device Functions 3 12-81
GreResetBounds 12-82
GreGetBoundsData 12-83
GreAccumulateBounds 12-84
GreGetCodePage 12-85
GreSetCodePage 12-86
GreLockDevice 12-87
GreUnlockDevice 12-88
GreGetLineOrigin 12-89
GreSetLineOrigin 12-90
Color Functions 12-92
GreQueryColorData 12-94
GreQueryLogColorTable 12-95
GreCreateLogColorTable 12-96
GreRealizeColorTable 12-98
GreUnrealizeColorTable 12-99
GreQueryRealColors 12-100
GreQueryNearestColor 12-101
GreQueryColorIndex 12-102
GreQueryRGBColor 12-103
Query Functions 12-104
GreQueryDeviceBitmaps 12-105
GreQueryDeviceCaps 12-106
GreQueryDevResource 12-108
GreQueryHardcopyCaps 12-113
GreEscape 12-115
Defined Escape Codes 12-116
Ranges for Additional Escape Codes 12-116
GreEscape DEVESC_ABORTDOC 12-117
GreEscape DEVESC_BREAK_EXTRA 12-118
GreEscape DEVESC_CHAR_EXTRA 12-119
GreEscape DEVESC_DRAFTMODE 12-120
GreEscape DEVESC_ENDDOC 12-121
GreEscape DEVESC_FLUSHOUTPUT 12-122
GreEscape DEVESC_GETCP 12-123
GreEscape DEVESC_GETSCALINGFACTOR 12-124

GreEscape DEVESC_NEWFRAME 12-125
GreEscape DEVESC_NEXTBAND 12-126
GreEscape DEVESC_QUERYESCSUPPORT 12-127
GreEscape DEVESC_QUERYVIOCELLSIZES 12-128
GreEscape DEVESC_RAWDATA 12-129
GreEscape DEVESC_STARTDOC 12-130
GreEscape DEVESC_STD_JOURNAL 12-131

Chapter 13. Mandatory Functions for Display Devices 13-1
AVIO Functions 13-2
GreCharRect 13-5
GreCharStr 13-6
GreScrollRect 13-7
GreUpdateCursor 13-8
GreDeviceSetAVIOFont 13-9
Miscellaneous Screen Functions 13-10
GreSaveScreenBits 13-11
GreRestoreScreenBits 13-12
GreDeviceInvalidateVisRegion 13-14
GreDeviceSetCursor 13-15
GreSetColorCursor 13-16
GreGetPickWindow 13-17
GreSetPickWindow 13-18
GreDeath 13-19
GreResurrection 13-20
GreGetDCOrigin 13-21
GreDeviceSetDCOrigin 13-22

Chapter 14. Simulated Functions 14-1
Arc Functions 14-2
GreGetArcParameters 14-3
GreSetArcParameters 14-4
GreArc 14-5
GrePartialArc 14-6
GreFullArcInterior 14-8
GreFullArcBoundary 14-10
GreFullArcBoth 14-12
GreBoxInterior 14-14
GreBoxBoundary 14-16
GreBoxBoth 14-18
GrePolyFillet 14-20
GrePolyFilletSharp 14-21
GrePolySpline 14-23
Area and Path Functions 14-24
GreAreaSetAttributes 14-25
GreBeginArea 14-26
GreEndArea 14-28
GreBeginPath 14-30
GreEndPath 14-32
GreCloseFigure 14-33
GreFillPath 14-34
GreOutlinePath 14-36
GreModifyPath 14-37
GreStrokePath 14-39
GreSelectClipPath 14-41
GreSavePath 14-43
GreRestorePath 14-44
Region Functions 14-45
GreGetRegionBox 14-46
GreGetRegionRects 14-47
GreOffsetRegion 14-49
GrePtInRegion 14-50
GreRectInRegion 14-51

GreCreateRectRegion 14-52
GreDestroyRegion 14-53
GreSetRectRegion 14-54
GreCombineRegion 14-55
GreCombineRectRegion 14-56
GreCombineShortLineRegion 14-57
GreEqualRegion 14-58
GrePaintRegion 14-59
Clip Functions 14-60
GreGetClipBox 14-61
GreGetClipRects 14-62
GreOffsetClipRegion 14-64
GrePtVisible 14-65
GreRectVisible 14-66
GreQueryClipRegion 14-67
GreSelectClipRegion 14-68
GreIntersectClipRectangle 14-69
GreExcludeClipRectangle 14-70
GreSetXFormRect 14-71
GreSaveRegion 14-72
GreRestoreRegion 14-73
GreSelectPathRegion 14-74
GreRegionSelectBitmap 14-75
GreCopyClipRegion 14-76
GreSetupDC 14-77
Transform Functions 14-78
GreGetPageUnits 14-82
GreSetPageUnits 14-83
GreGetModelXform 14-85
GreSetModelXform 14-86
GreGetWindowViewportXform 14-87
GreSetWindowViewportXform 14-88
GreGetGlobalViewingXform 14-89
GreSetGlobalViewingXform 14-90
GreSaveXformData 14-91
GreRestoreXformData 14-92
GreGetPageViewport 14-93
GreSetPageViewport 14-94
GreGetGraphicsField 14-95
GreSetGraphicsField 14-96
GreGetViewingLimits 14-97
GreSetViewingLimits 14-98
GreConvert 14-99
GreSaveXform 14-100
GreRestoreXform 14-101
GreMultiplyXforms 14-102
GreConvertWithMatrix 14-103

Chapter 15. Graphics Engine Internal Functions 15-1
Device Context Functions 15-2
GreOpenDC 15-3
GreCloseDC 15-6
GreResetDC 15-7
GreGetHandle 15-9
GreSetHandle 15-10
GreGetProcessControl 15-11
GreSetProcessControl 15-12
GreSaveDC 15-13
GreRestoreDC 15-14
GreQueryEngineVersion 15-15
Device Support Functions 15-16
GreCreateBitmap 15-17
GreDeleteBitmap 15-19

GreSelectBitmap 15-20
GreGetBitmapParameters 15-21
GreGetBitmapDimension 15-22
GreSetBitmapDimension 15-23
GreGetAttributes 15-24
GreSetAttributes 15-25
GreSetGlobalAttribute 15-27
GreGetDefaultAttributes 15-28
GreSetDefaultAttributes 15-29
GreGetDefaultArcParameters 15-31
GreSetDefaultArcParameters 15-32
GreGetDefaultViewingLimits 15-33
GreSetDefaultViewingLimits 15-34
GreInitializeAttributes 15-35
Set ID Functions 15-36
GreDeleteSetId 15-37
GreQueryNumberSetIds 15-38
GreQuerySetIds 15-39
LCID Functions 15-40
GreQueryBitmapHandle 15-41
GreSetBitmapID 15-42
GreCopyDCLoadData 15-43
Font Functions 15-45
GreQueryLogicalFont 15-46
GreCreateLogicalFont 15-47
GreLoadFont 15-50
GreUnLoadFont 15-51
GreQueryFonts 15-52
GreQueryFontAttributes 15-53
GreQueryCodePageVector 15-54
GreQueryFontFileDescriptions 15-55
Journaling Functions 15-56
GreCreateJournalFile 15-57
GreDeleteJournalFile 15-58
GreStartJournalFile 15-60
GreStopJournalFile 15-62
GrePlayJournalFile 15-63
GreOpenJournalFile 15-65

Chapter 16. System Functions 16-1
FSRSemCheck 16-2
FSRSemEnter 16-3
FSRSemExit 16-5
FSRSemLeave 16-6
GetDriverInfo 16-7
InnerGreEntry 16-8
SetDriverInfo 16-9
SSAllocHuge 16-10
SSAllocSeg 16-11
SSFreeSeg 16-12
VioGetPSAddress 16-13
VisRegionCallBack 16-14
WinSetErrorInfo 16-15

Chapter 17. Presentation Manager Spooler 17-1
Spooler Components 17-1
Spool File Creation 17-1
Spooler Functions 17-3
SplStdClose 17-4
SplStdDelete 17-5
SplStdGetBits 17-6
SplStdStart 17-7
SplStdStop 17-8

SplStdOpen 17-9
SplStdQueryLength 17-10
The Queue Manager (Print Manager) 17-11
SplQmAbort 17-12
SplQmAbortDoc 17-13
SplQmClose 17-14
SplQmEndDoc 17-15
SplQmOpen 17-16
SplQmStartDoc 17-17
SplQmWrite 17-18
SplMessageBox 17-19
The Queue Processor (Queue Driver) 17-20
How a Queue Processor Prints 17-20
SplQpClose 17-21
SplQpControl 17-22
SplQpInstall 17-23
SplQpOpen 17-24
SplQpPrint 17-26
SplQpQueryDt 17-27

Chapter 18. File System Emulation 18-1
PrtAbort 18-2
PrtClose 18-3
PrtDevIOCtl 18-4
PrtOpen 18-5
PrtWrite 18-6

Appendix A. Syntax Conventions A-1
Parameter Names A-1

Appendix B. Initialization File B-1
PM_SPOOLER_DD B-1
PM_SPOOLER_PORT B-1
PM_SPOOLER_QP B-2

Glossary X-1

Index X-11

Chapter 10. Presentation Drivers Overview

Presentation drivers are special purpose I/O routines, operating with I/O privilege at privilege level 2 (ring 2). These drivers provide the link between the internal interface to presentation devices and the OS/2 interfaces to the kernel device drivers at ring 0 and the screen (see Figure 10-1 on page 10-2).

The code for presentation drivers is held in dynamic link libraries that, when loaded and enabled, process the function calls passed through an internal dispatch table. These libraries are identified as presentation drivers by the file-name extension .DRV.

Support is provided in the *Programming Tools and Information* package for writing source code in either C or Macro Assembler. This support comprises a set of header files, included through PMDDI.H or PMDDI.INC, that define the functions, structures, and constants used on the internal interface to the presentation driver.

In common with other components of OS/2, the presentation driver architecture ensures that:

- Once loaded, the driver can be enabled for use by multiple applications or processes.
- Once enabled, the driver can support multiple instances of a device context for the owning application or process.

Display Devices

When the Presentation Manager is initialized, the presentation driver for the attached display is loaded and enabled. This driver has direct access to the video hardware. The function calls passed to the driver are processed and passed to the adapter interface.

Printers and Plotters

For printers and plotters, the presentation driver is loaded in response to an application or process calling DevOpenDC. Upon receipt of this call, the Presentation Manager looks to see if the required driver is loaded and if it is able to handle the new DC (device context): if both are true, the driver is enabled for the new DC; if either is false, the required driver is loaded and enabled.

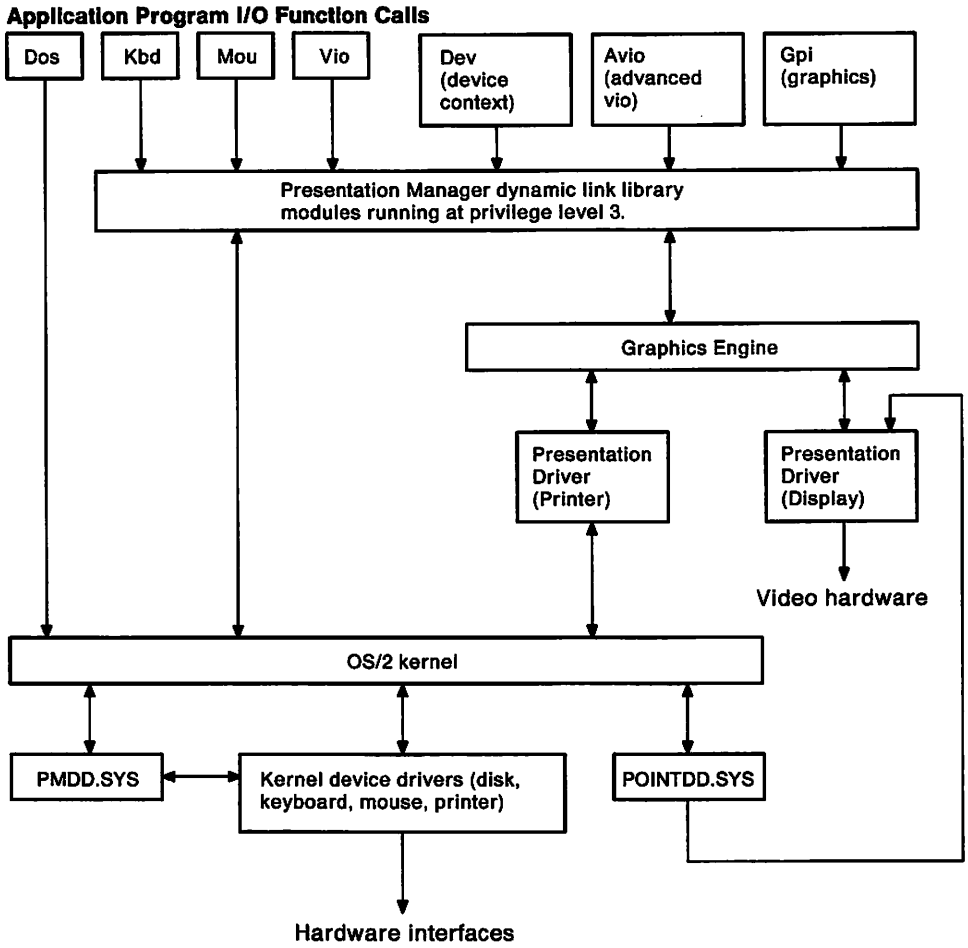


Figure 10-1. Presentation Drivers. Conceptual view of presentation drivers in the flow of control from an application program to the printer and the display screen.

Functions

Presentation Driver Interface

The internal presentation driver interface (PDI) comprises a set of graphics engine (Gre...) functions that are called through a dispatch table. This table is an array of pointers to function-handling routines; the low-order byte of the function number identifies the member of the array that contains the pointer for the function. The functions called through the dispatch table fall into three main groups:

1. Functions that all presentation drivers must support. (See Chapter 12, "Mandatory Functions for All Devices.")
2. Functions that must be supported by presentation drivers for display devices. (See Chapter 13, "Mandatory Functions for Display Devices.")
3. Functions that are supported by simulations in the graphics engine. (See Chapter 14, "Simulated Functions.")

Each instance of a loaded presentation driver is given a copy of the default dispatch table. The enable routine in the driver modifies this copy so that, for those functions that are supported in the driver, the pointers address the driver's function-handling routines.

Definitions of the functions and structures are included through the files PMDDI.H and PMDDI.INC. These definitions include symbolic names for the Gre... function numbers.

Dynamic Link Library Functions

Functions exported and imported by a dynamic link library are identified in the library module definition file. These provide links between libraries and subsystems; for example, the components of Presentation Manager need to call an enable entry point in the presentation driver and the presentation driver needs access to the simulated functions in the graphics engine.

For a typical module definition file, see the .DEF file in the sample printer presentation driver. For details of how to develop a dynamic link library (DLL), see the *Programming Guide* and *Building Programs*.

Note: The initialization routine for a dynamic link library, including presentation drivers, must be compiled to run at ring 3 (privilege level 3).

Exported Functions

Presentation Drivers: Dynamic link libraries for presentation drivers must export the appropriate set (see comments in list) of the following entry points:

```
EXPORTS
   OS2_PM_DRV_ENABLE         /* all drivers          */
   OS2_PM_DRV_DEVMODE        /* not display drivers  */
   OS2_PM_DRV_DEVICENAMES    /* not display drivers  */
   MoveCursor                /* display drivers only */
```

In addition to the entry points listed above, printer and plotter presentation drivers should export entry points for the routines that handle dialogs with the user. For details, see Chapter 11, "Exported Entry Points."

Graphics Engine Functions: The graphics engine exports its own set of entry points. Those that are significant to the presentation driver are:

```
EXPORTS
   InnerGreEntry    @3

   GETDRIVERINFO    @30
   SETDRIVERINFO    @31
```

InnerGreEntry The entry point for all simulations and functions supported by the graphic engine. For details, see Chapter 14, "Simulated Functions" and Chapter 15, "Graphics Engine Internal Functions."

GETDRIVERINFO Used by the presentation driver to get the instance pointer (pInstance) for a specified device context or to get a pointer to the bit map header for a specified bit map.

SETDRIVERINFO Used by the presentation driver to set a specified value in the instance pointer of a specified device context.

Notes:

1. GETDRIVERINFO and SETDRIVERINFO are discussed in Chapter 16, "System Functions."
2. Instance pointers (pInstance) and data are discussed under "Enable subfunction 05H — EnableDeviceContext" on page 11-12.

Imported Functions

Presentation driver libraries must call the InnerGreEntry entry point in the graphics engine to gain access to the internal engine functions. Typically you would import the entry point by ordinal (PMGRE.3) and assign a meaningful name, or names to it. For example, to import the journal functions, your module definition file would contain:

```
IMPORTS
        GreCreateJournal =PMGRE.3
        GreDeleteJournal =PMGRE.3
        GreStartJournal  =PMGRE.3
        GreStopJournal   =PMGRE.3
        GrePlayJournal   =PMGRE.3
        GreOpenJournal   =PMGRE.3
```

To call a Gre... function that is supported as a simulation or internal function in the graphics engine, you call the imported entry point with the Gre... function parameters; the pInstance parameter should be null. For example, to call GreCreateJournalFile using the name assigned in the module definition file, you would use:

```
result = GreCreateJournal (pszFileName, flOption, cSize, 0L, NGreCreatJournalFile );
```

Notes:

1. NGreCreateJournalFile is defined in the header file PMDDIM.H.
2. GreCreateJournalFile is described on page 15-57.
3. Simulations (that is, presentation driver interface functions that are supported by handling routines in the graphics engine) can also be called at the addresses given in the default dispatch table; use the addresses contained in the dispatch table that is passed to the presentation driver at enable time.

Device Context

Device contexts provide the mechanism that the application program uses to write output data to devices. The application, or one of its processes, opens a device context (using DevOpenDC), associates a presentation space to the DC, and writes or draws in that space. Each DevOpenDC creates an instance of a DC and that instance is destroyed when the application closes the device context.

The created DC is seen internally as a dispatch table. Calls from the application program to the DC are passed, as one or more internal Gre... calls, via the dispatch table to the handling routines in the presentation driver for the DC.

Each instance of a DC has:

Device type
Data type
Instance data
Program stack.

Device Types

The type of device that is being opened is passed to the presentation driver when the device context is enabled:

OD_DIRECT The driver processes the Gre... calls to generate device-specific output data. Printer and plotter drivers use the file system Prt... interface to pass the output to the kernel device driver; display drivers use the adapter interface (for example, the IBM Personal System/2 Display Adapter 8514/A adapter interface).

OD_QUEUED The driver opens a spool file and uses the spooler Spl... interface to send output to that file. For spooled output, the driver has to take account of the DC data type, see Data Types below.

OD_INFO The driver does not generate any output. Information device contexts are used to retrieve information; all Gre... function calls passed to the driver are processed as if the type was OD_DIRECT so that the system can query statistics such as font metrics and boundaries.

OD_MEMORY The driver processes the output data as for an OD_DIRECT device type except that the output is written to a bit map that is compatible with the physical device. (The application program creates the bit map and associates it to the device context.)

Data Types

For queued devices, the presentation driver needs to support the PM_Q_STD data type as defined by Presentation Manager. Support for other data types is optional.

The data type that is to be used is passed to the presentation driver when the device context is enabled:

PM_Q_STD The driver uses the spooler to create a file. When the application calls DevCloseDC, the driver stops the file and uses the spooler SplQM... interface to pass the file to the queue manager.

PM_Q_RAW The driver processes the Gre... calls to generate device-specific output data. This data is written, using the spooler SplQm... interface, to a spool file.

Instance Data

For every instance of a device context, the system has a doubleword that is reserved for use by the presentation driver. Typically, this doubleword is used by the driver to hold a pointer to information about the current state of the device context. This pointer is returned to the system by the driver when the device context is enabled; on subsequent calls through the dispatch table, the pointer is passed back to the driver as a parameter, pInstance, on the program stack.

Program Stack

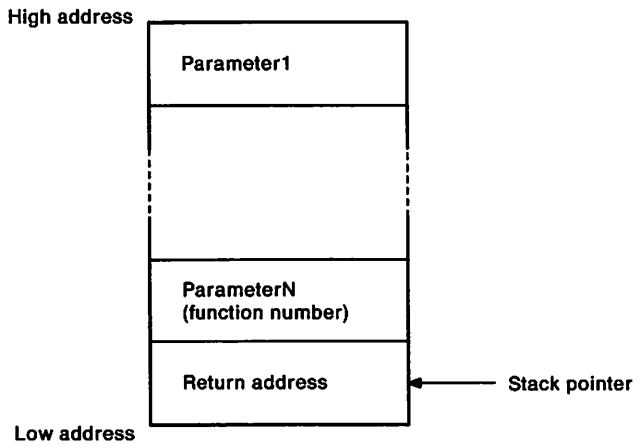
Presentation device drivers can assume that a stack of 500 bytes is available for use when a function call is passed to the driver. If it needs more than 500 bytes, the driver should allocate its own stack space, switch to that stack on entry, and switch back to the original stack on exit.

Function calls to the driver use Pascal calling conventions. That is, parameters are pushed to the stack in the same order as they are in the call statement. The last parameter pushed by the calling routine is the function number; from this number the driver can determine the number and meaning of the other parameters.

Calling Conventions

Presentation drivers interact with both the external API and the internal interface to the graphics engine and dispatch table. Parameters are passed as 32-bit values on the stack using the Pascal convention; that is, they are pushed to the stack in the order that they appear in the statement. The hardware architecture locates the conceptual "bottom" of the stack at the high-order address of the stack space; when data is pushed to the stack, the stack pointer is decremented so that upon completion the pointer addresses the last item of data.

At entry to the handling routine, the stack contains a frame of 32-bit parameters and a return address:



The internal interface differs from the external API in that the last parameter pushed to the stack is a function number. This is because internal calls are passed to common entry points for dispatch to the function-handling routines.

Function Number and Command Flags

The first double word pulled from the stack contains two fields:

Function number (low-order word)	Identifies a specific function. (Files PMDDIM.H and PMDDIM.INC define symbolic names for the Gre... function numbers.)
Command flags (high-order word)	See below.

The flags in the 16-bit command flags field tell the driver of operations that need to be done while it processes the function:

COM_DRAW (bit 0)

If the COM_DRAW flag is set, the driver must draw the output of the function at the device. (When the flag is not set, the driver does not draw the output; but the function must still be processed to update internal data such as current position and, if specified, boundary calculations.)

COM_BOUND (bit 1)

If the COM_BOUND flag is set, the driver must calculate the bounding rectangle for the output of the specified function. Upon completion the driver calls its own GreAccumulateBounds routine to accumulate the bounding rectangle (that is, GPI_BOUNDS in model space coordinates).

Note: All presentation drivers must be able to calculate bounds on any figure that they can draw.

COM_CORR (bit 2)

This flag is significant only for display drivers. If the COM_CORR flag is set, the driver must determine whether the output of the specified function intersects the pick window. The result, true or false, is passed back to the caller in the return code for the called function; for details, see the function descriptions in Chapter 12 and Chapter 14.

COM_ALT_BOUND (bit 3)

This flag is significant only for display drivers. It indicates that the driver should accumulate USER_BOUNDS in screen coordinates.

Note: USER bounds are those used by the window manager; presentation drivers for hard-copy devices do not accumulate USER bounds.

COM_AREA (bit 4)

If the COM_AREA flag is set, the function call is part of an area. Calls to functions that define an area (for example, GrePolyLine and GreSetCurrentPosition) or are invalid in an area definition should be passed back to the graphics engine for processing by the default handler.

Note: If all functions in the area component have been hooked by the presentation driver, it is not necessary to pass back functions received with COM_AREA set.

COM_PATH (bit 5)
This flag is similar to the COM_AREA flag. Calls to functions that define a path (for example, GrePolyLine and GreSetCurrentPosition) or are invalid in an path definition should be passed back to the graphics engine for processing by the default handler.

Note: If all functions in the path component have been hooked by the presentation driver, it is not necessary to pass back functions received with COM_PATH set.

COM_TRANSFORM (bit 6)
If the COM_TRANSFORM flag is set, any coordinates given for the specified function must be transformed, using GreConvert, to device coordinates.

COM_RECORDING (bit 7)
This flag should be ignored.

COM_DEVICE (bit 8)
If the COM_DEVICE flag is set, the presentation driver must process the function; the driver should not pass the function to the graphics engine.

Notes:

1. COM_DRAW, COM_BOUND, COM_CORR, COM_ALT_BOUND, COM_AREA, and COM_PATH apply only to drawing functions; they should be ignored by all other functions.
2. The remaining bits of the command flags field are not defined; their value should be ignored.

Design Considerations

Sample Presentation Driver

The Programmer's Tools for OS/2 1.2 has source code for a printer presentation driver, PMPRT.DRV, in the TOOLKIT12\PD\SAMPLES\PMPRT directory. This source code shows a typical implementation of a basic printer presentation driver and references to the source code modules have been included in this book.

General Considerations

Return Codes: The presentation driver must always return a full 32-bit (LONG) value. For example, BOOLEAN TRUE and FALSE are defined as:

```
#define TRUE   (1L);
#define FALSE  (0L);
```

Coordinate Values: All coordinates are passed to the presentation driver as 32-bit values. Unless stated otherwise, these values represent world coordinates. The graphics engine function GreConvert can be called to convert coordinates from one type to another. Coordinates must be converted back to world coordinates before returning.

Screen coordinates are device coordinates to which the DC origin has been added.

Transform Matrix Values: Transform matrix elements are represented in fixed point notation; that is as a 16-bit signed integer and a 16-bit fractional part. These precision limits apply during graphics engine matrix multiplication for all initial, intermediate, and final matrix element values.

Angles: Angles are passed as signed 32-bit numbers: zero refers to the direction of the positive x axis; 2^{31} represents 360°. Positive values represent counterclockwise angles from the positive x axis.

Bounds Computation: All presentation drivers must accumulate bounds for unclipped primitives. Application bounds (COM_BOUND) are accumulated in model space and user bounds (COM_ALT_BOUND) in device coordinate space.

Closing Figures in Areas and Paths: The graphics engine generates closure lines for figures within areas and paths, unless the presentation driver has opted to hook all the path and area functions. In this instance, the presentation driver is responsible for closing any figures. For further details see "Area and Path Functions" on page 14-24.

Correlation: Correlation must be performed by all presentation drivers for display devices, in page coordinates on fully clipped primitives. Correlation is not required for printer and plotter devices.

Correlating on areas is particularly complex. This is because GreSetCurrentPosition and GreEndArea generate a closure line when the current position is not at the start of the current closed figure. This closure line can cause a correlation hit. Also, the area interior itself can cause a correlation hit that must be reported on the GreEndArea order.

The lines (and arcs, full arcs, boxes, fillets) defining the area boundary can cause a correlation hit if the area is specified with boundary. This hit must be reported when the function is issued. This means that some other work must be done in addition to journaling the calls that define the area boundary.

Positions Within Text Functions: Any text function that uses positions takes the position from the base line of the text box. Descenders, such as the tail of a lowercase y, are expressed as a negative value relative to the base line.

Clipping: The presentation driver must perform clipping for drawing and text functions except for GreDrawLinesInPath, and GrePolyShortLine. Clipping for these two exceptions is done by the graphics engine. The minimum requirement is to be able to render each primitive clipped to a single rectangle, and to clip each rectangle in turn. The rectangles can be enumerated using GreGetClipRects (see page 14-62).

Note: Rectangles may not always be valid, see "Drawing to Display Devices."

Interrupts

Presentation drivers should never use the CLI and SLI Macro Assembler instructions. These instructions may interfere with some of the base OS/2 system operations.

Drawing to Display Devices

Because changes on the screen can affect more than one DC, the graphics engine notifies the display driver when a change occurs. Notification is through a call to the GreInvalidateVisRegion (see page 13-14) routine in the display driver. This call identifies the affected DCs and supplies a pointer (pInstance) to the instance data of each DC. The handling routine for GreInvalidateVisRegion should set a flag, such as HDC_IS_DIRTY, in the instance data for all identified DCs.

All routines that draw on the screen should test the HDC_IS_DIRTY flag and, if it is set, call the graphics engine's VisRegionCallBack (see page 16-14) function before drawing the visible region.

Exit List Processing

Note: The exit routine in the presentation driver must be compiled to run at ring 3.

At enable time the presentation driver must place an entry in the exit list for the application or process that opens the DC. This entry is a pointer identifying the routine in the presentation driver that releases all resources owned by the DC.

For presentation drivers, their exit routine would check the Fast-safe RAM (FSR) semaphores and release those that are owned by the dying process. ExitList processing is the only occasion when the driver accesses the system fields of the driver's FSRSEM structures; the driver checks the usProcessID field and, if the ID is that of the dying process, it calls FSRSemExit to release the semaphore.

Note: When writing a presentation driver, you must consider what would happen if another thread of the process were to terminate.

Allocating Memory

Presentation drivers have two ways of allocating and managing segments of memory:

1. Using the Dos... functions such as DosAllocSeg and DosAllocHuge.
2. Using the SS... functions described in Chapter 16, System Functions.

Drivers for display devices or drivers that wish to share objects such as bit maps and regions, should always use the SS... functions to allocate memory for these objects. Memory allocated through calls to these functions is shared memory controlled by the segment selector component of the graphics engine; ownership of the memory can be transferred or, when the owning DC ceases to exist, marked as having no owner.

Protecting Objects

Objects such as a DC_ heap and bit maps must be protected to prevent their use by other tasks and processes. Because a presentation driver can be entered at any Gre... routine, one of the first tasks of a routine is to lock the DC and any objects used by the routine. This locking is done with Fast-Safe RAM (FSR) semaphores using calls to the FSRSemCheck, FSRSemEnter, FSRSemExit, and FSRSemLeave functions; these are system functions and they are described in Chapter 16, "System Functions."

A process which attempts to use a locked object should return an error such as PMERR_HDC_BUSY.

Protecting the DC

Although a DC is owned by a single application or process, the owner may access the DC through multiple threads. The presentation driver must provide a mechanism whereby a DC can register that it is busy and block access from other threads. In its simplest form, this would be done by EnterDriver and LeaveDriver routines that are called at the start and end of each function-handling routine in the driver.

A typical EnterDriver routine for a display driver would be:

```
enter_driver()
{
      do {
                    /* Lock the DC for the exclusive use of */
                    /* the current thread.                  */
          SemEnter(Device);

                    /* Some functions do not pass a DC handle. */
          if (hdc == NULL)
              return(SUCCESS);

                    /* Check the validity of the passed DC handle */
          if (hdc == ERROR) {
              WinSetErrorInfo (SEVERITY_ERROR, PMERR_INV_HDC);
              SemLeave(Device);
              return(ERROR);
          }

                    /* The DC region must be validated    */
                    /* before the driver can draw into    */
                    /* it.                                */
          if (hdc_is_not_dirty) /* Test the HDC_IS_DIRTY flag */
              return(SUCCESS);

                     /* If the flag is set, the DC  */
                     /* region must be recalculated*/
                     /* by the system.*/

          SemLeave(Device);      /* Unlock the DC */

                     /* Call back to the engine to */
                     /* force DC calculation.      */
          VisRegionCallBack(hdc);
                    /* Loop back to reset the lock */
                    /* and recheck. */
      } while (TRUE);
}
```

Bit Map Simulation

Presentation drivers for monochrome raster devices (such as the IBM 4201 Proprinter) can use the system's display driver (DISPLAY.DLL) to draw the page image on a bit map. This technique reduces the amount of code in the printer driver and ensures that all devices use the same drawing algorithms.

To use the display driver, the printer presentation driver has to open and manage a display DC. This must be done without invoking the graphics engine; the printer driver has to act as if it were the engine. When an application opens a printer DC, the enable subfunctions in the printer driver must issue the appropriate calls to the enable subfunctions in the display driver.

The sample printer presentation driver uses bit-map simulation. Initialization is done by the printer driver's enable subfunctions in module N1ENABLE.C:

FillLogicalDeviceBlock

1. Loads the display driver and gets the address of its enable entry point.
2. Saves a copy of the default dispatch table to pass to the display driver's FillLogicalDeviceBlock routine.
3. Calls the display driver's enable entry point with the parameters set for FillLogicalDeviceBlock and passing a pointer to the saved default dispatch table. The display driver will initialize this dispatch table and it can then be used to pass Gre... calls to the display DC.

FillPhysicalDeviceBlock

1. Saves, in the printer's physical device block, the address of the display driver's enable entry point.
2. Calls the display driver's enable entry point with the parameters set for FillPhysicalDeviceBlock and OD_MEMORY device type.
3. Saves the value returned from the display driver's FillPhysicalDeviceBlock routine. (This value, ulStateInfo, needs to be passed back to the display driver's EnableDeviceContext routine.)

 Note: In some source code you might see the name pDCI used for the ulStateInfo parameter.

EnableDeviceContext

1. Calls the display driver's enable entry point with the parameters set for EnableDeviceContext. The DENPARAMS structure passed to the EnableDeviceContext routine contains the same DC handle (ulHDC) that was received by the printer driver.
2. Saves the value returned from the display driver's FillPhysicalDeviceBlock routine. (This value, pInstance, needs to be passed back to the display driver on every call through the dispatch table.)

 Note: In source code you might see pInstance referred to as the 'magic cookie'.

CompleteOpenDC

Calls the display driver's enable entry point with the parameters set for CompleteOpenDC.

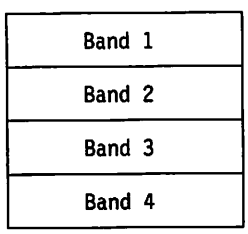
The situation now is that any Gre... calls from the system to the DC enter the printer DC through its dispatch table. Function-handling routines in the printer driver monitor the incoming calls and redirect those calls that affect the image through the display DC's dispatch table. (The sample presentation driver manages dispatching in modules T0DPATCH.ASM and T1DPATCH.C.)

When a GreEscape call for DEVESC_NEWFRAME or DEVESC_ENDDOC is detected, the printer presentation driver transfers the bit-map bits from the display DC to a local buffer and sends them as scan lines or bands to the kernel device driver.

Banding

Banding is another technique that is available to presentation drivers for raster devices, it is used to reach a balance between memory requirements and performance. This technique uses the graphics engine's journaling functions to save and replay a journal file of the Gre... calls for a whole page.

The driver handles the output page as a number of bands and creates a bit map big enough for one band:



The DC origin of the bit map is manipulated so that it relates to each band in turn. And the device driver replays the journal file as many times as necessary to write into each band. Note that band 1 can be written whilst the Gre... calls are being journaled.

Each page of output is handled as a separate entity: the GreEscape routine for DEVESC_STARTDOC opens a journal file for the first page and registers it in the DC instance data. When GreEscape DEVESC_NEWFRAME or DEVESC_ENDDOC is received the driver writes the bands and closes the journal; if the escape code was DEVESC_NEWFRAME, the GreEscape routine opens and registers the journal file for the next page. Figure 10-2 gives an overview of what the presentation driver needs to do.

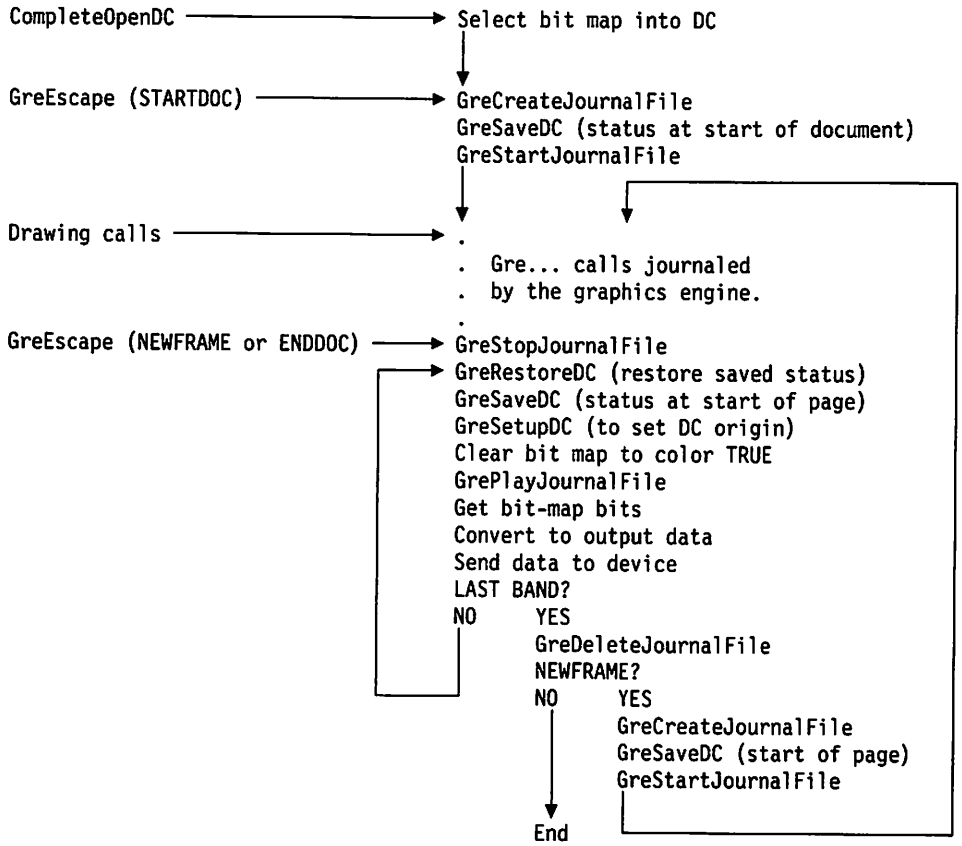


Figure 10-2. Banding

Error Strategy

Presentation drivers should support the error strategy implemented by the Presentation Manager. When an error occurs, the driver should call WinSetErrorInfo (see page 16-15) to log the appropriate error code and set the return code to show that an error was detected.

The component that implements a function must provide error checking for the environment, objects, and resources associated with it. Your presentation driver needs to cater for:

- Fail-safe on routines that set attributes and transformation values. Any routine that changes attributes or transformation values must be able to restore the initial values if an error occurs during the change.
- Full error checking on symbol sets, fonts, bit maps, and regions.
- For segment drawing, drawing primitives, and primitive attributes in draw mode, unchecked parameters are passed directly to the graphics engine or the presentation driver. When one of these functions is hooked by the presentation driver, the handling routine must do the necessary error checking and log any errors, or reset any invalid values to their defaults, as appropriate.

- For any function with coordinates as parameters, the presentation driver must check that the values passed are valid. When an invalid coordinate is detected, the handling routine must log an error, or use a default coordinate value, as appropriate.

For any defined error, the application sees the same error code, regardless of whether the error was logged by the GPI, graphics engine, or presentation driver.

Severity

Four severity levels are defined for error messages: warning; error; severe error; unrecoverable error.

Warning: The function detected a problem but took some remedial action and was able to complete successfully.

Error: The function detected a problem for which no sensible remedial action is possible. The function is not executed and the system remains at the same state as when the function was requested.

Severe Error: The function detected a problem from which the system cannot reestablish its state. The function has partially executed and the application must now make some corrective action to restore the system to some known state.

Unrecoverable Error: The function detected an error from which it is impossible for the system to reestablish the state that it held at the time that the function was called. It is also impossible for the application to restore the system to some known state.

Presentation Manager Error Codes

Error codes are defined in the header file PMGPI.H. These codes fall into two groups; general and specific. General error codes, appropriate to many Gre... functions, include:

Error	Must be logged by
PMERR_COORDINATE_OVERFLOW	Functions requiring matrix computation.
PMERR_INSUFFICIENT_MEMORY	Functions resulting in memory allocation.
PMERR_INV_HBITMAP	Functions with hbm as an explicit, or implicit, parameter.
PMERR_INV_HRGN	Functions with hrgn as an explicit, or implicit, parameter.
PMERR_INV_COORDINATE	Functions with coordinate parameters.
PMERR_INV_IN_AREA	Functions that are invalid inside an open area bracket.
PMERR_BASE_ERROR	Functions that directly, or indirectly, issue DOS calls.
PMERR_DEV_FUNC_NOT_INSTALLED	All functions that must be supported by the presentation driver.

Specific error codes, appropriate to individual Gre... functions, are listed in the descriptions of each call.

Chapter 11. Exported Entry Points

This chapter describes the entry points that must be exported by a presentation driver dynamic link library:

```
EXPORTS
   OS2_PM_DRV_DEVMODE        /* not display drivers  */
   OS2_PM_DRV_DEVICENAMES    /* not display drivers  */
   OS2_PM_DRV_ENABLE         /* all drivers          */
   MoveCursor                /* display drivers only */

   DialogProc      @2         /* not display drivers */
```

DialogProc is exported by ordinal; the entry point is used by the Presentation Manager to manage the dialog initiated by the OS2_PM_DRV_DEVMODE routine. See OS2_PM_DRV_DEVMODE in this chapter.

OS2_PM_DRV_DEVMODE

Note: The OS2_PM_DRV_DEVMODE handling routine must be compiled to run at ring 3 (privilege level 3).

The device modes entry point is exported, by the presentation driver, as OS2_PM_DRV_DEVMODE to support the DevPostDeviceModes function at the API. In the presentation driver, the handling routine generates a DRIVDATA structure that defines the current setting of device options or the setting of job properties that identify the options that should be set when the job is printed (see "Remarks" below). All presentation drivers for hard-copy devices must contain a handling routine for OS2_PM_DRV_DEVMODE.

Flags passed to the handling routine indicate whether or not the routine should read driver data from or write it to the OS2SYS.INI file; but note that when pDriverData is a valid value, the DRIVDATA structure must be passed back to the calling application. One flag bit (bit 1) determines whether the handling routine should establish a dialog with the user to complete the structure or whether it should use the information read from OS2SYS.INI. Another flag bit (bit 0) tells the handling routine whether or not it should update the OS2SYS.INI file.

Applications such as the Presentation Manager control panel call DevPostDeviceModes to configure the device. Note that such applications would usually call this function twice: first with a null value for pDriverData to query the length of the driver's DRIVDATA structure; and then with a valid pointer to get the data.

The syntax used by the Presentation Manager to call the device modes routine in the presentation driver is:

```
LONG EXPENTRY OS2_PM_DRV_DEVMODES ( pDriverData, pszDriverName, pszDeviceName, pszName, flFlags )

PDRIVDATA pDriverData;
PSZ       pszDriverName;
PSZ       pszDeviceName;
PSZ       pszName;
ULONG     flFlags;
```

Note: LONG, EXPENTRY (pascal far), PDRIVDATA (DRIVDATA far *), and PSZ (char far *) are defined in OS2DEF.H, which is included through the header file OS2.H.

Stack Frame

At entry to the device modes routine, the stack frame contains:

pDriverData	(null or pointer to DRIVDATA structure, see below)
pszDriverName	(pointer to a string containing the name of the device driver)
pszDeviceName	(pointer to a string containing the device name, such as IBM 4201, as defined by the presentation driver)
pszName	(null or pointer to a string containing the device name, such as PRINTER1, as defined by the user through the Presentation Manager control panel; a null pointer or a null string are both valid conditions for this parameter)
flFlags	(option flags, see below)

pDriverData Null or pointer to memory location for returned DRIVDATA structure:

cb	Number of bytes in the structure.
lVersion	Version number of the presentation driver. (Subsequently used by the driver to verify its entry in the .INI file.)
szDeviceName[32]	Device name.
abGeneralData	Driver-specific data for device options and job properties. See "Remarks" below.

If pDriverData is null, the handling routine in the presentation driver must return the length of the driver's DRIVDATA structure.

flFlags Bits 1 and 0 identify the action that should be taken by the presentation driver:

00	Display a dialog for job properties and return the DRIVDATA structure, do not update OS2SYS.INI.
01	Display a dialog for device options and job properties, update the OS2SYS.INI file, and return the DRIVDATA structure.
10	Return the DRIVDATA structure with device options and job properties copied from OS2SYS.INI, do not display a dialog.
11	Reserved.

All other flags are reserved.

Return Code

The handling routine in the presentation driver should return a LONG integer. Valid values are:

−1	Error.
0	There are no settable options.
>0	The number of bytes in the DRIVDATA structure.

Remarks

Details of device options and job properties are stored as a set of flags in the array abGeneralData: do not store pointers in this array, they might not be valid when you get them back. This array is driver-specific; you determine what flags are needed to fully exploit the capabilities of the device and where those flags are in the array.

When designing dialogs you should consider the purpose and usability of the dialog. The value of bit 0 in the flFlags parameter shows the purpose of the dialog:

0 The calling program wants the user to select properties for a specific job: draft or letter quality; size, style, and color of the default font; and so on. The source for the default value of the properties and device options is determined by the pszName parameter:

- If pszName points to a valid string, the handling routine searches the PM_SPOOLER_DD section of OS2SYS.INI for the abGeneralData associated with the name. (See Appendix B, Initialization File for details of the PM_SPOOLER_DD section of OS2SYS.INI.)
- If pszName is null or points to a null string, the handling routine uses the values from abGeneralData in the DRIVDATA structure addressed by pDriverData.

In both cases, the initial default values are used if the handling routine cannot find a valid abGeneralData array.

1 The calling program wants the user to identify the current settings of device options and select the default settings for job properties. This would usually require two dialogs: one to identify options such as the paper size currently in the device and details of any memory or font cartridges that are installed; the other to establish a set of default job properties.

Note: Information concerning the design of dialogs and menus is given in *IBM Systems Application Architecture Common User Access: Advanced Interface Design Guide.*

OS2_PM_DRV_DEVICENAMES

Note: The OS2_PM_DRV_DEVICE_NAMES handling routine must be compiled to run at ring 3 (privilege level 3).

The device names entry point is exported as OS2_PM_DRV_DEVICENAMES by the presentation driver to support the DevQueryDeviceNames function at the API. Applications such as the Presentation Manager control panel call DevQueryDeviceNames to determine the device names and descriptions and the data types that the presentation driver supports. Presentation drivers for hard-copy devices must contain a handling routine for OS2_PM_DRV_DEVICENAMES.

Applications would usually call this function twice: first with a null value for cNames and cDataTypes to query the number of names and data types; and then, having allocated the arrays, with valid values to get the data.

If the value of cNames, at the location addressed by pcNames, is null, the handling routine must update cNames to the actual count of names. If cNames has a valid value, the routine must write device names and device descriptions into the arrays addressed by paDeviceName and paDeviceDesc. Similarly for cDataTypes, the handling routine either writes a valid value in cDataTypes or data-type names in the array addressed by paDataType. Note that when writing to an array, the routine should not write past the end of the array as defined by the associated count.

The syntax used by the Presentation Manager to call the device names routine in the presentation driver is:

```
LONG EXPENTRY OS2_PM_DRV_DEVICENAMES ( pszDriverName, pcNames, paDeviceName, paDeviceDesc,
                                       pcDataTypes, paDataType, lReserved1, lReserved2 )

PSZ          pszDriverName;
PLONG        pcNames;
PSTR32       paDeviceName;
PSTR64       paDeviceDesc;
PLONG        pcDataTypes;
PSZ          paDataType;
ULONG        lReserved1;
ULONG        lReserved2;
```

Note: LONG, EXPENTRY (pascal far), and PSZ (char far *) are defined in file OS2DEF.H. PSTR16, PSTR32, and PSTR64 are defined in file PMDEV.H as far pointers to fixed-length character arrays. Both files are included through the header file OS2.H.

Stack Frame

At entry to the device names function, the stack frame contains:

pszDriverName	(pointer to a string containing the name of the device driver)
pcNames	(pointer to count of fields, cNames, in DeviceName and DeviceDesc arrays)
paDeviceName	(pointer to DeviceName array: char[cNames,32]. Device names are null terminated strings such as 'IBM 4201'.)
paDeviceDesc	(pointer to DeviceDesc array: char[cNames,64]. Device descriptions are null terminated strings such as 'IBM 4201 Proprinter'.)
pcDataTypes	(pointer to count of fields, cDataTypes, in DataType array)
paDataType	(pointer to Type array: char[cTypes,16])
lReserved1	(reserved)
lReserved2	(reserved)

Return Code

The handling routine in the presentation driver should return a LONG integer. Valid values are:

−1 Successful.
0 Error.

Note: The above values are not a typographic error; the system really does expect the successful and error return codes from OS2_PM_DRV_DEVICENAMES to be the opposite of those from OS2_PM_DRV_DEVMODE and the Enable subfunctions.

OS2_PM_DRV_ENABLE

The enable entry point is exported as OS2_PM_DRV_ENABLE by the presentation driver. The enable routine in the driver supports a set of subfunctions that enable/disable the environment for a device context owned by a specific application or process. When a device context is opened or closed, the Presentation Manager issues a series of calls to subfunctions at the enable entry point. These calls initialize the presentation driver, the physical device, and the device context.

The syntax used by the Presentation Manager to call the enable routine is:

```
LONG EXPENTRY OS2_PM_DRV_ENABLE ( Subfunc, Param1, Param2 )

ULONG   Subfunc;
ULONG   Param1;
ULONG   Param2;
```

Note: LONG, ULONG (unsigned long) and EXPENTRY (pascal far) are defined in OS2DEF.H which is included through the header file OS2.H.

Stack Frame

At entry to the enable routine, the stack frame contains:

Subfunction	(32-bit value identifying the subfunction)
Param1	(first parameter)
Param2	(second parameter)

Subfunctions

Presentation drivers have to support the following enable subfunctions:

01H	FillLogicalDeviceBlock
02H	FillPhysicalDeviceBlock
04H	DisablePhysicalDeviceBlock
05H	EnableDeviceContext
06H	DisableDeviceContext
07H	SaveDCState
08H	RestoreDCState
09H	ResetDCState
0AH	CompleteOpenDC
0BH	BeginCloseDC.

Note: The enable function should return –1 (ERROR_MINUS) for subfunction 03H and numbers greater than 0BH.

Opening a Device Context

Device contexts are opened in response to an application or process calling the DevOpenDC API function. On receiving this call, Presentation Manager loads the presentation driver, if it is not already present and able to support the device context, and issues a series of calls to the enable entry point.

Initially, when the presentation driver has not been enabled for use by the calling application or process, the driver receives the series of calls shown in Figure 11-1. When the driver has been enabled for an application or process, the sequence of calls to enable additional device contexts does not include FillLogicalDeviceBlock and, depending upon a requirement that the driver posts in the initial enable sequence, includes or excludes FillPhysicalDeviceBlock.

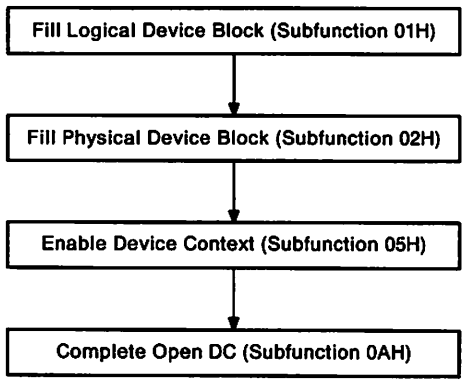


Figure 11-1. Enabling a presentation driver.

This subfunction is called on the first occasion that a specific application or process opens a device context that uses the called presentation driver; it is not called when the same application or process opens additional DCs with the same presentation driver. However, the graphics engine can call this subfunction at other times and the call should always be honored: that is, if the value of pDispatchTable is nonzero, the handling routine must initialize the dispatch table.

The major tasks that the handling routine must implement are:

1. Add an entry to the DosExitList to ensure that any allocated resources are freed when the owning application or process terminates.
2. Save those pointers in the dispatch table that will be needed to pass hooked functions back to the graphics engine. (A typical example is GreCharString: this has to be hooked by the driver but can be passed back to the default handling routine.)
3. Initialize the dispatch table. That is, modify the table so that the entries for functions hooked by the presentation driver contain pointers to the driver's handling routines.
4. Set flags to indicate how future DevOpenDC calls to this device should be handled.

In some typical drivers, the handling routine for FillLogicalDeviceBlock allocates global heap space for use by the device contexts. The memory for this heap space is obtained by calls to the SSAllocHuge and SSAllocSeg functions described in Chapter 16, System Functions. This global heap space is available to all instances of a DC that are opened by the application or process for which the presentation driver was enabled.

Stack Frame

ulSubfunction 01H

pParam1 Long pointer to the following structure:

- **ulVersion** Version number, in binary coded decimal (BCD), of the graphic engine.
- **cTableSize** The number of entries in the dispatch table. The driver should not replace pointers past the end of the table.

pParam2 Long pointer to the following structure:

- **pfsFlags** Pointer to a word of logical device flags. Flag bits 0 and 2 apply to the presentation driver and should be set on or off, as appropriate, by the driver. All other flags are reserved for system use and must not be modified.
 - **Bit 0** Set on if each DC for this driver requires its own physical device block.
 - **Bit 2** Set on if the szDeviceName and pszDriverName fields of a DevOpenDC call for this device should be ignored. Setting bit 2 on indicates that the presentation driver supports only one physical device in one configuration; for example, the display driver.
- **pDispatchTable** Pointer to the presentation driver's copy of the dispatch table. This table is an array of 32-bit pointers to system-default function handling routines; the low-order byte of a function number identifies the offset to the relevant pointer.

Enable subfunction 01H – FillLogicalDeviceBlock

Return Codes

The handling routine should return a LONG integer. Valid values are:

–1	Error.
0	Successful.

The FillPhysicalDeviceBlock subfunction is always called in the initial set of calls to the enable function for a specific application or process; it is only called when additional device contexts are enabled if the value of flag bit 0, as set in the initial call to FillLogicalDeviceBlock, shows that each device context requires its own physical device block. However, the graphics engine can call this subfunction at other times: if it is called more than once when individual DCs do not require separate physical device blocks, the handling routine does nothing and returns the existing ulStateInfo handle or pointer from the DC instance data.

The physical device block is located in the driver's global heap. To initialize the block, the driver uses:

- Default values set by the driver
- Values read from the OS2.INI file
- Values from the DEVOPENSTRUC structure.

A typical physical device block is shown under Remarks below.

Stack Frame

ulSubfunction 02H

pParam1 Long pointer to an extended DEVOPENSTRUC structure (the structure is extended by adding a DENPARAMS structure to give three additional fields, ulStateInfo, ulType, and ulHDC):

pszLogAddress Pointer to the logical address of the device. (For example, 'LPT1'.)

pszDriverName Pointer to name of the presentation driver. (For example, 'IBM4201'.)

pdriv Pointer to DRIVDATA structure. This structure contains data generated by the presentation driver during the dialog that set device modes. See "OS2_PM_DRV_DEVMODE" on page 11-2.

- **cb** Number of bytes in the structure.
- **lVersion** Version number.
- **szDeviceName[32]** Device name.
- **abGeneralData** Driver-specific data.

pszDataType Pointer to the name of the queue file data type:

PM_Q_STD
PM_Q_RAW

pszComment Pointer to a file description that the spooler could use in messages displayed to the user.

pszQueueProcName Pointer to name of queue processor.

pszQueueProcParams Pointer to a string of queue processor parameters.

pszSpoolerParams Pointer to a string of spooler parameters separated by one or more blanks. Valid parameters are:

FORM = aaa
Identifies the form name for the print job; multiple names are separated by commas (aaa,bbb,ccc). If this parameter is not present, the job is printed on the current form.

Form names are defined by the presentation driver; valid names are those that would be returned from a call to the driver's GreQueryHardcopyCaps handling routine.

PRTY = n
Identifies the priority for the print job. The priority can be any value from 1 through 99 (1 is lowest priority).

pszNetworkParams Pointer to a string of networking parameters.

Enable subfunction 02H – FillPhysicalDeviceBlock

ulStateInfo Reserved; this field does not contain meaningful information at this time.

ulType DC type (for example, OD_QUEUED).

ulHDC DC handle.

ulParam2 Reserved.

Return Codes

The handling routine should return a LONG integer. Valid values are:

–1 Error.

other Handle or pointer, ulStateInfo, to the physical device block. (This pointer will be passed back to the presentation driver in subsequent calls to the Enable subfunctions 04H and 05H.)

Note: You will see the name pDCI used in place of ulStateInfo in some source code. See "Remarks" below.

Remarks

Physical device blocks hold information about the driver and device that is the same for every instance of a device context. The design of the driver determines what information is held in the physical device block. A typical printer physical device block is given below.

```
typedef struct
{
    Dptr        PDBDriverName;          /* String for Driver Name      */
    Dptr        PDBOutputName;          /* String for Output Name      */
    unsigned    PDBHandle;              /* Handle for DOS Device       */
    Word        PDBOutputType;          /* Type of output; STD/ESC/RAW */
    DeviceSemaphoreTable PDBDevice;
                                        /* pointer to device table     */
    Byte        PDBScratch[DCT_MAX_SIMPLE];
                                        /* Scratch Pad for Printer     */
    DDTType     DDT;                    /* copy of the DDT to be used  */
    Word        RasterMode;             /* raster type used            */
    Word        PrintQuality;           /* draft or LQ printing        */
    Word        Orientation;            /* portrait or landscape       */
    Dword       Version;                /* driver version number       */
    DevRect     FormClipRegion;         /* clip region for current form*/
    Byte        DeviceName[32];         /* model number, and so on.    */
} PDBIType;
```

In the sample printer presentation driver, FillPhysicalDeviceBlock is handled by module N1ENABLE.C. This driver requires a separate physical device block for each instance of a DC and the physical device block contains information that other drivers would put in the DC instance data.

Note: The driver flags its requirement for separate physical device blocks while it is handling FillLogicalDeviceBlock (see page 11-7).

Enable subfunction 04H – DisablePhysicalDeviceBlock

This subfunction in the presentation driver is called by the system to disable the specified device and to free any associated memory.

Presentation drivers for the primary display device should return a value of 0 without taking any action.

Note: The system will never call this subfunction in the presentation driver for a hard-copy device if the driver uses one physical device block to support multiple DCs. (Drivers notify the system of this capability by not setting bit 0 of the logical device flags returned to the system from the FillLogicalDeviceBlock subroutine.)

Stack Frame

ulSubfunction 04H

ulParam1 Handle or pointer, ulStateInfo, to physical device block.

pParam2 Not used.

Return Codes

The handling routine should return a LONG integer. Valid values are:

–1 Error.
0 Successful.

Enable subfunction 05H – EnableDeviceContext

In response to this call, the driver reserves the memory it needs to support this instance of a DC and initializes the instance data. The value of the return code from this function, if it is not −1, is saved by the system and passed back to the driver on all future calls to this DC instance; this value is expected to be a pointer or handle to the instance data. Instance data is described under "Remarks" below.

The handling routine should initialize relevant fields in the instance data to their default values; for example, it would call WinQueryProcessCP to get the initial code page ID. Note that this call is not available at the external API, its definition is:

```
USHORT APIENTRY WinQueryProcessCP( VOID );
```

Stack Frame

ulSubfunction 05H

pParam1 Long pointer to a DENPARAMS structure:

ulStateInfo
Value returned by FillPhysicalDeviceBlock.

ulType
Type of device context. Defined values are:
OD_QUEUED
OD_DIRECT
OD_INFO
OD_MEMORY.

For details, see DevOpenDC in the *Programming Reference*.

ulHDC
Device context handle.

pParam2 Not used.

Return Codes

The handling routine should return a LONG integer. Valid values are:

−1 Error.
other Pointer, pInstance, to the DC instance data.

Remarks

Instance data refers to information, such as the name of a spool file and whether a bit map has been created, that applies to a specific instance of a device context. Global data, that applies to all instances of a DC using the same presentation driver, is held in the physical device block (PDB); a pointer to this block is passed, in the DENPARMS structure, to the EnableDeviceContext routine to be included in the instance data.

In the sample printer presentation driver, the instance data is contiguous with the state information date in the physical device block and is defined in module EPGLOBAL.H. That sample uses hDDC as the name for the pointer to the instance data. The acronym DDC is derived from Device's Drawing Context.

A typical example of instance data is given below.

```
typedef struct dci
    {
        struct dci far *DCINextEntry;  /* next instance                */
        Dword           DCIDCType;     /* DC type 0 -> 8               */
        lpBitmap        DCISelBitmap;  /* selected bit map             */
        HHEAP           DCIHeapHandle; /* heap handle                  */
        Word            DCIHeapSel;    /* heap selector                */
        WcsPoint        DCICurrPosWcs; /* current position - World     */
        DevPoint        DCICurrPosDev; /* current position - Device    */
        WcsPoint        DCIPattOriWcs; /* pattern origin - World       */
        DevPoint        DCIPattOriDev; /* pattern origin - Device      */
        PDCHARBUNDLE    DCICurTxtAts;  /* Current Text Attrs Bundle    */
        PDLINEBUNDLE    DCICurLinAts;  /* Current Line Attrs Bundle    */
        PDAREABUNDLE    DCICurPtnAts;  /* Current Pattern Attrs Bundle*/
        PDIMAGEBUNDLE   DCICurImgAts;  /* Current Image Attrs Bundle   */
        PDMARKERBUNDLE  DCICurMrkAts;  /* Current Marker Attrs Bundle  */
        lpFontTable     DCIFontTabHead;/* Head of font table chain     */
        Boolean         DCIPairKerning;/* Enabled\Disabled state       */
        enum BandState  DCIBandState;  /* State of Band processing     */
        BandType        DCICurrBand;   /* Current Band                 */
        Word            DCIBandLowY;   /* Lowest Y Coord from Bands    */
        Word            DCIJournLowY;  /* Lowest Y Coord in Journal    */
        Dword           DCIJournHandle;/* Handle for Journal           */
        char            DCIJrnNameff12";/* Journal name XXXXXXXX.XXX   */
        Word            DCIDosJrnHnd;  /* Jrn handle for Dos writes    */
        SplHandle       DCISplHandle;  /* handle to spool file         */
        lpColorTable    DCIColorTable; /* color table                  */
        Word            DCIColTabSize; /* color table size             */
        Word            DCIColFormat;  /* format of color table        */
        Word            DCILowIndex;   /* lowest index in table        */
        Word            DCIHighIndex;  /* highest index in table       */
        lpPDBI          DCIPdbInstance;/* PDB Instance Data            */
        lpClipRectangle DCIClipRects;  /* Clip Rectangles              */
        Word            DCIClipNum;    /* Number of Clip Rectangles    */
        Word            DCIClipSize;   /* Number of Clip Rectangles    */
                                       /* that storage has been        */
                                       /* allocated for                */
        Word            DCILineTypMask;/* Mask used for line types     */
        Word            StyleNumber;   /* line style state and mask    */
        Dword      far *DCIDocName;    /* DC document name pointer     */
        Word            DCIPrintInfo;  /* printing info for print band*/
        Boolean         DCIStartedDoc; /* TRUE if a doc is started     */
        long int        DCISaveCount;  /* number of saved DC states    */
        Dword           DCICodePage;   /* currently selected codepage */
        DevRect         DCIBounds;     /* currently accumulated bounds*/
        Boolean         DCIDefBounds;  /* are bounds default           */
        Word            ProcessID;     /* current process ID           */
        SPoint          DCIOrigin;     /* current DC origin            */
        lpDownloadFont  DCIDownloadFont;/* download font list          */
        Word            DCIDownloadNum;/* number of download fonts     */
        Word            DCIDownloadMax;/* current length of font list */
        Boolean         DCIDefaultClip;/* using the default clip rect */
        Word            DCIBandClipNum;/* Number of Clip Rectangles in*/
                                       /* current band.                */
        BandType        NextBand;      /* band used by NextBand        */
    } DCIType;
```

Enable subfunction 06H – DisableDeviceContext

This function is called when a device context is about to be deleted. In response, the driver must release any memory and other resources that it has allocated for the DC. The presentation driver should use the DC instance data to identify this memory.

Stack Frame

ulSubfunction 06H

pParam1 Pointer, pInstance, to the DC instance data.

pParam2 Not used.

Return Codes

The handling routine should return a LONG integer. Valid values are:

–1 Error.
0 Successful.

Requests the driver to save all of the information that it has about the device context. The state of a DC may be saved multiple times in *last in first out* (LIFO) order. The routine should return an error code if there is not enough memory available to save the state.

Note: The handling routine needs to keep a count of the number of saved states.

Stack Frame

ulSubfunction 07H

pParam1 Pointer, pInstance, to the DC instance data.

pParam2 Not used.

Return Codes

The handling routine should return a LONG integer. Valid values are:

–1 Error.
0 Successful.

Enable subfunction 08H – RestoreDCState

Restores a specific DC state from the saved DC states. An index to the required state is supplied as parameter 2: the driver should return an error if the index is zero or if it specifies a value that does not identify a saved state.

Display presentation drivers to be need careful when handling this call; some of the data stored in the DC instance data must match the data held by the window manager. The handling routine for RestoreDCState should not restore:

DC origin
USER bounds
Cached clipping rectangles
The HDC_IS_DIRTY flag.

Stack Frame

ulSubfunction 08H

pParam1 Pointer, pInstance, to the DC instance data.

lParam2 Identifies which of the saved states are to be restored.

Positive numbers indicate the specific state counting from the first saved state; one being the first, two the second, and so on. All saved states following the one being restored should be discarded. Preceding states remain saved.

Negative numbers indicate the specific state counting from the last saved state; thus, a value of minus two shows that the last state should be discarded and the state before that be restored.

Return Codes

The handling routine should return a LONG integer. Valid values are:

–1 Error.
0 Successful.

Resets the device context to its original initialized state. The driver should delete all fonts, patterns, and paths, and reset all attributes to their default values. Note that when resources are not owned by the driver, the driver should save the relevant values so that they are available to reset the device context. A typical example is when the default font is a graphics engine font, in this case the driver would save the font flags and address passed in the first call to GreDeviceSetAttributes.

The visible region and the DC origin are not affected by this function.

Stack Frame

ulSubfunction 09H

pParam1 Pointer, pInstance, to the DC instance data.

ulParam2 Reserved.

Return Codes

The handling routine should return a LONG integer. Valid values are:

–1 Error.
0 Successful.

Enable subfunction 0AH – CompleteOpenDC

This function is called upon completion of the DevOpenDC process to tell the driver that the device context now has access to the graphics engine. Presentation drivers for the primary display device should return successful without taking any action. For other devices, the handling routine in the driver completes the initialization process for any resources, such as bit maps, that are obtained via calls to the graphics engine.

Presentation drivers for hard-copy devices should not use the CompleteOpenDC routine to open resources such as spool files or journal files. If these resources are required, they should be opened in response to a call to GreEscape with DEVESC_STARTDOC (see page 12-130). Such drivers would set a flag in the instance data to show that DEVESC_STARTDOC had been received and would not process any output until that flag had been set.

Note: The presentation driver should lock and unlock resources such as bit maps and device contexts to prevent simultaneous use of the resource by two threads belonging to the same process. Typically this would be done by setting a semaphore or some other form of "busy" flag for the resource.

Stack Frame

ulSubfunction 0AH

hdcParam1 Device context handle.

pParam2 Pointer, pInstance, to the DC instance data.

Return Codes

The handling routine should return a LONG integer. Valid values are:

–1 Error.
0 Successful.

This function is called to inform the driver that the device context is being closed; this is the last call made to the driver before it loses access to the graphics engine. For the primary display device, the driver should return successful without taking any action. For other devices, the handling routine in the driver has to close any resources, such as journal files and bit maps, that it owns.

Presentation drivers for hard-copy devices should not use the BeginCloseDC routine to complete tasks such as writing spool files. The tasks should be completed in response to a call to GreEscape with DEVESC_ENDDOC (see page 12-121). The DEVESC_ENDDOC routine should reset the DEVESC_STARTDOC flag in the instance data and the BeginCloseDC routine should check that the flag is reset before taking any action.

Stack Frame

ulSubfunction 0BH

hdcParam1 Device context handle.

pParam2 Pointer, pInstance, to the DC instance data.

Return Codes

The handling routine should return a LONG integer. Valid values are:

–1 Error.
0 Successful.

MoveCursor

Presentation drivers for displays must export an entry point for a MoveCursor routine. This routine supports calls from the system timer, at interrupt time, from the POINTDD.SYS driver, and from pointer calls made by an application. The strategy for MoveCursor is that the pointer is checked and, if necessary, redrawn or excluded at timed intervals.

Some constraints are placed upon the MoveCursor routine to accommodate system requirements:

- The word immediately preceding the MoveCursor entry point must be reserved for use by the system and the MoveCursor routine.
- The routine must use the segment addresses passed to it in the DS and CS registers.
- The routine must call all support routines using a near call.
- The routine must not load any segment register with anything other the original value of the register (registers DS, CS, and SS).

The word immediately preceding the MoveCursor routine is used to pass the selector for the segment that contains all the data needed to draw the cursor or pointer. The PMDD.SYS kernel device driver creates a privilege level 0 alias for the specified segment and, when MoveCursor is called at interrupt time, passes the alias to the handling routine in the AX register.

At entry to the MoveCursor routine, the stack contains the x and y device coordinates of the new location for the cursor hot spot:

```
void MoveCursor(abs_x,abs_y)
SWORD abs_x;                    /* x coordinate of cursor */
SWORD abs_y;                    /* y coordinate of cursor */
{

}
```

Regular timer interrupts give the presentation driver an opportunity to check that the pointer is valid. That is:

- Have new x,y coordinates been set?
- Is the pointer excluded because of a drawing operation and, if so, has that operation completed?
- Is the pointer currently visible when it should be excluded because it is in an area that is being redrawn?

When MoveCursor is called at interrupt time the coordinates on the stack are both 8000H, and the form of the first instruction in the MoveCursor routine determines whether the routine is entered at the MoveCursor address or at MoveCursor+3. If the first instruction is of the form 'mov ax,0xxxxh', the routine is entered at MoveCursor+3 to preserve the content of the AX register.

At the end of the MoveCursor routine there should be a check to see if a new location was given for the pointer while it was being drawn. If the pointer has moved again, it must be drawn at the new location (or be excluded because it has moved into the protection rectangle). This implies that the routine needs to track both real(x,y) and pointer(x,y).

Programming Considerations

Typical cursors are an arrow or cross with an action point, the hot spot, at the point of the arrow and the center of the cross. When the driver draws a cursor, the origin of the image must be offset to place the action point at the required (x,y) position; the required offset is specified in the call to GreSetCursor.

Because the cursor entry point may be called at various times from many different places; the cursor routine should use semaphores to protect itself (protection is the responsibility of the presentation driver). Similarly, because cursor drawing can be a time-consuming operation, the driver must also protect itself against reentrancy.

The presentation driver must resolve all interactions between cursor drawing at interrupt time and access to video hardware. While in the background, the driver should not draw any cursor image.

Care needs to be taken when the display is a buffered device and the cursor is drawn into a bit map in the buffer. In this instance the driver needs to delete the cursor and exclude it when a draw operation occurs at the cursor location. To do this, the driver needs to do a hit test for each output operation to see if the cursor location is in the drawing area and to set a protection rectangle that is used to exclude the cursor.

Chapter 12. Mandatory Functions for All Devices

This chapter describes those functions, called through the dispatch table, which must be supported for all devices by handling routines in the presentation driver.

The dispatch table contains the address of each function that the presentation driver can hook. Initially, when the presentation driver is first loaded, a copy of the default dispatch table is passed to the driver. The driver's Enable routine modifies this copy so that the entries for functions supported in the driver point to the handling routines in the driver. Entries to the table that are to be modified can first be saved by the presentation driver in case they are subsequently needed.

The original, saved, table entries and those that are not modified point to the engine-simulation routines for the functions concerned.

Functions listed in the dispatch table fall into two categories; those that the presentation driver *must* support (*mandatory functions*) and those that are supported by the graphics engine but can optionally be hooked by the presentation driver.

This chapter provides descriptions of the mandatory functions. Each description shows what the handling routine is expected to do, the parameters passed to the routine, and the values that the routine should return. The functions are grouped according to the conditional include sections of the header file PMDDIM.H:

- Line functions (INCL_GRE_LINES)
- Scan functions (INCL_GRE_SCANS)
- Bit-map functions (INCL_GRE_BITMAPS)
- String functions (INCL_GRE_STRINGS)
- Marker functions (INCL_GRE_MARKERS)
- Attribute functions (INCL_GRE_DEVMISC1)
- Miscellaneous device functions (INCL_GRE_DEVMISC2)
- Miscellaneous device functions (INCL_GRE_DEVMISC3)
- Color table functions (INCL_GRE_COLORTABLE)
- Device functions (INCL_GRE_DEVICE)
- Escape function (INCL_GRE_DEVICE).

Additional functions, which must be supported by presentation drivers for display devices are described in Chapter 13, "Mandatory Functions for Display Devices." Presentation drivers for printer and plotter devices must provide some support for these display-device functions. This can be a common entry point that returns success and posts a warning (PMERR_DEV_FUNC_NOT_INSTALLED).

The level of support that you provide in your handling routine for a particular function, depends on the device, and the level of support that you intend to provide for it. The handling routine must, at least, indicate a successful completion.

Line Functions

- GreGetCurrentPosition
- GreSetCurrentPosition
- GrePolyLine
- GreDrawLinesInPath
- GrePolyShortLine
- GrePolyScanLine.

Line and Curve Styling

The style, or type, of a line determines whether it is drawn solid, as a series of dots, dashes, dashes and dots, or as any combination of dots and dashes. This is determined by the usType parameter in the LINEBUNDLE structure, (see "Line (Pen) Attributes" on page 12-52) and the style ratio.

When a styled line is drawn, the style is copied into a style mask, which is stored as a byte in the instance data structure of the device context. This structure is described under "Enable subfunction 05H — EnableDeviceContext" on page 11-12. At the same time, mask position and error-term bytes, which are used to calculate the styling process, are set to zero in the instance data structure. x and y step values, which can be returned by GreGetStyleRatio or set to your own defaults, are also stored in the same structure. When GreDrawLinesInPath is used for drawing, the style state from the path structure is used instead of that from the device context instance (DCI) structure.

The line is then drawn by determining its track between the start and end points. For each pel on the track, the style mask is consulted. When the most significant bit of the style mask is 1, the pel is drawn. If this bit is 0, the pel is not drawn. A step value from the style ratio is then added to the error-term byte and the next pel on the track is considered. When the line is styled y-major, it is the y-step that is added for each movement. Otherwise it is the x-step that is added for each movement in the x-direction. When the error-term byte overflows, the style mask is rotated one bit to the left and the mask position byte is incremented by one. The style mask is reset when the value of the mask position reaches a value of eight. Only the lower three bits of the mask-position byte are required for this, the rest are reserved for use by the engine. This algorithm is demonstrated in the example shown on page 12-90. When the line type is LINETYPE_ALTERNATE, the GreGetStyleRatio function must return zero for pRatio, regardless of the actual value. This allows the driver to recognize the line style as alternate and to set the x and y-step values to 256. The result is that the style mask is rotated after every pel.

For a line (AB), the change in the style state can be expressed as:

```
z = max(Cx * {Bx - Ax} , Cy * {By - Ay})
```

Where Cx and Cy are the step values for the x and y directions, respectively. The change in the style state is determined by the mask position and error term bytes:

```
mask position = (old mask position) + (z/256)) MOD 8
error term = (old error term + z) MOD 256
```

When the mask position and error term bytes are combined into a single word (w), so that the mask position is the high order byte, at point A, the style state at B is:

```
mask position = (high-order byte of (z + w)) MOD 8
error term = (low-order byte of (z + w))
```

When drawing PolyShortLines, the engine approximates the shortline with a line between its end points. If the engine determines that the line is y-major it sets the PSL_YMAJOR bit of the style field in the shortline header structure, otherwise, the bit is set to zero and the line is styled as x-major. This is done because it is not always possible to apply the maximum metric to a shortline with enough accuracy. For instance, when a driver has a y-step that is greater than its x-step and this situation exists:

```
z = max(Cx * {Bx - Ax} , Cy * {By - Ay}) = Cy * {By - Ay}
```

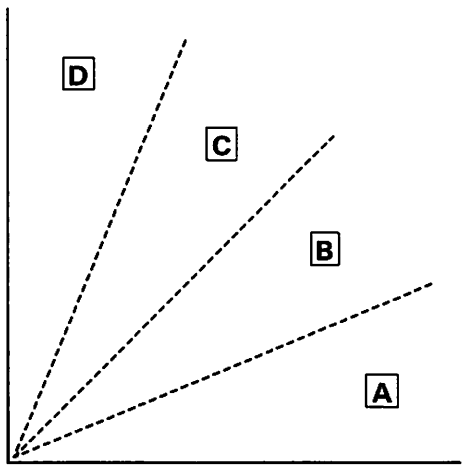
and

```
{Bx - Ax} >  {By - Ay}
```

- The shortline has at least one horizontal section.
- The line is classed y-major.

If this line is now clipped to the horizontal section only, the presentation driver will label it as x-major. The result is that the last pel of the line in the clip region may not match the corresponding pel in the unclipped version.

Special consideration must be made when styling lines on EGA devices. This is because the difference in the x and y-pel aspect can cause vertical and horizontal lines, drawn with the same attributes, to appear different on the screen. A line has two components that affect its direction: a *run* component, and a *side* component. The run component defines the general direction of the line, the side component is the direction of the push that has to be applied to the run to make the line hit its specified end point. Run and side operate in the horizontal, vertical, and diagonal directions. When the maximum metric is applied, the presentation driver for the EGA device must map each step value twice. The first mapping determines the major direction of styling, the second determines the run and side steps.



Direction			Style steps	
Region	Run	Side	Run	Side
A	Horizontal	Diagonal	Cx'	Cx' + Cy'
B	Diagonal	Horizontal	Cx' + Cy'	Cx'
C	Diagonal	Vertical	Cx' + Cy'	Cy'
D	Vertical	Diagonal	Cy'	Cx' + Cy'

Figure 12-1. Line Styling with Run and Side

In the figure, a line drawn from the bottom-left corner to a point in region A is drawn with horizontal run steps. Between each run, a side step is drawn diagonally northeast. After each step, the run or side value is added to the error term, as appropriate, to control the rotation of the style mask. When the line is styled x-major, `Cx' = Cx, Cy' = 0`. When the line is styled y-major, `Cx' = 0, Cy' = Cy`.

First and Last Pel Considerations

It is the responsibility of the presentation driver to ensure that a series of line, arc, and fillet orders all join up correctly, including the on/off counts defined by the current line attributes. As an example, when drawing connected lines (PolyLines) the handling routine must not draw the first pel of the second, and subsequent, lines.

Typically, the presentation driver maintains a flag in the DC instance data structure to indicate whether the first pel of a line is drawn. This flag should be set by GreSetCurrentPosition and cleared by any subsequent drawing primitive.

To ensure that a figure is closed correctly, GreCloseFigure does not draw the last pel in the closure line.

Some orders are defined to be *move* type operations. A move causes three things to happen:

- The line style sequence is reset.
- The next line, arc, fillet, or partial arc primitive, is drawn with first and last pel (subject to the line style sequence).
- In an area, if the current figure is not closed (the current device coordinate position is not the same as the start device coordinate position), an implicit closure line is drawn to close it.

Subsequent start line, arc, fillet, and partial arc primitives, are drawn to include the last, but not the first, pel (subject to the line style sequence).

Any full arc, box, or pie slice, drawn with a boundary (that is any closed figure), is drawn with its boundary complete (no missing pels), and with the line pattern sequence honored around all the parts of its boundary. Such closed figures are not considered to be move type operations so as to allow construction of complex area boundaries.

Move type operations are:

- SetCurrentPosition.
- Any Set that changes, or might change, the transform from world coordinate space to device coordinates, for instance SetModelTransform or SetWindow/ViewportTransform.
- Any Set that changes, or might change, the current clipping. For example SetViewingLimits.

Note: You will need a different set of rules if you want to construct a boundary for scan-line area filling. As an example, you can choose to ignore line style, and draw all lines solid, with first pel off, last pel on (see "GreGetLineOrigin" on page 12-89). This boundary is, of course, different from the boundary that is drawn on the screen after the interior is filled.

Functions, such as GrePolyLine, GreArc, and GrePolyFillet, that are preceded by a move operation are drawn with the first pel on and the last pel set off.

```
(BOOL) GreGetCurrentPosition (hdc, pptPosition, pInstance, lFunction)
```

Takes the current (x,y) position, in world coordinates, from the DC instance data structure and stores it at the location addressed by pptPosition.

Support

This function must be supported by all presentation drivers.

The sample driver handles this call by bit-map simulation. The call parameters are passed unchanged to the display driver's dispatch table.

Stack Frame

(HDC)	hdc	(device-context handle)
(PPOINTL)	pptPosition	(pointer to current position)
(ULONG)	pInstance	(pointer to instance data)
(ULONG)	lFunction	(High-order word = Flags, Low-order word = NGreGetCurrentPosition)

Return Codes

On completion, the handling routine must return BOOLEAN (fSuccess):

TRUE Successful.
FALSE Error.

When an error is detected, the handling routine must call WinSetErrorInfo to post the condition. Error codes for conditions that the handling routine is expected to check include:

PMERR_DEV_FUNC_NOT_INSTALLED
PMERR_INV_HDC.

GreSetCurrentPosition

```
(BOOL) GreSetCurrentPosition (hdc, pptPosition, pInstance, lFunction)
```

Sets the current (x,y) position and resets the line-type sequence.

Typically, the handling routine also sets a flag in the DC instance data structure to indicate that the first pel of the next line must be drawn.

When the COM_AREA or COM_PATH flag is set, this function is part of an area or path definition. In either case the handling routine would usually pass the call back to the graphics engine for processing by the default handling routine. The call would not be passed back to the graphics engine if the presentation driver had hooked all of the area and path functions.

Support

This function must be supported by all presentation drivers.

The sample driver handles this call by bit-map simulation. The call parameters are passed unchanged to the display driver's dispatch table.

Stack Frame

(HDC)	hdc	(device-context handle)
(PPOINTL)	pptPosition	(pointer to new current position, see below)
(ULONG)	pInstance	(pointer to instance data)
(ULONG)	lFunction	(High-order word = Flags, Low-order word = NGreSetCurrentPosition)

pptPosition When COM_TRANSFORM is set, the current position is expressed in world coordinates, otherwise this value is passed in device coordinates. The handling routine must transform these values as appropriate. Typically, the presentation driver maintains the current position in both coordinate sets in the DC instance data structure.

Return Codes

On completion, the handling routine must return BOOLEAN (fSuccess):

TRUE Successful.
FALSE Error.

When an error is detected, the handling routine must call WinSetErrorInfo to post the condition. Error codes for conditions that the handling routine is expected to check include:

PMERR_BASE_ERROR
PMERR_PATH_LIMIT_EXCEEDED
PMERR_HDC_BUSY
PMERR_DEV_FUNC_NOT_INSTALLED
PMERR_COORDINATE_OVERFLOW
PMERR_INV_LENGTH_OR_COUNT
PMERR_INV_HDC
PMERR_INV_COORD_SPACE.

Remarks

When the current context is *in area*, a figure closure line is generated if necessary (which could cause a correlation hit to occur on an area boundary). The current position should only be correlated on, or merged into the bounds, or both, if it is actually used in a drawing primitive.

Thus, as an example, the sequence:

```
          ...
GreSetCurrentPosition ( hdc, p1 );
          ...
          ...
          ...
GreSetCurrentPosition ( hdc, p2 );
          ...
          ...
          ...
GrePolyLine ( hdc, to p3, n );
          ...
```

does not merge p1 into the bounds or correlate on it.

GrePolyLine

```
(LONG) GrePolyLine (hdc, paptPoint, cPoints, pInstance, lFunction)
```

Draws a sequence of one or more lines starting at the current (x,y) position. As each line is drawn, its end point becomes the start point for the next line. Upon completion, the current (x,y) position is the end point of the last line.

When the COM_AREA or COM_PATH flag is set, this function is part of an area or path definition. In either case the handling routine would usually pass the call back to the graphics engine for processing by the default handling routine. The call would not be passed back to the graphics engine if the presentation driver had hooked all of the area and path functions.

When this function is used to draw a closed figure, the handling routine must not draw the last point of the last (closure) line.

The handling routine can call back to the graphics engine to do any necessary clipping.

Support

For performance reasons, all presentation drivers should support this function when drawing a polyline to a single clipping rectangle. When the clip region is more complex, the handling routine can forward the call to the graphics engine using the pointer supplied in the dispatch table when the driver was enabled. The engine will clip each line and return them to the presentation driver as a call to GreDrawLinesInPath.

The sample driver handles this call by bit-map simulation. The call parameters are passed unchanged to the display driver's dispatch table.

Stack Frame

(HDC)	hdc	(device-context handle)
(PPOINTL)	paptPoint	(pointer to an array of (x,y) points, see below)
(LONG)	cPoints	(number of (x,y) pairs in points array)
(ULONG)	pInstance	(pointer to instance data)
(ULONG)	lFunction	(High-order word = Flags, Low-order word = NGrePolyLine)

cPoints When this is passed as zero, the handling routine takes no action, except to return successful.

paptPoint Pointer to an array of cPoints (x,y) pairs containing the (x,y) coordinates of the end points for the lines.

Return Codes

On completion, the handling routine must return an integer (cHits) indicating, where appropriate, whether correlation hits were detected:

GPI_OK Successful.
GPI_HITS Successful with correlate hit (returned by display drivers when the correlate flag is on and a hit is detected).
GPI_ERROR Error.

When an error is detected, the handling routine must call WinSetErrorInfo to post the condition. Error codes for conditions that the handling routine is expected to check include:

PMERR_BASE_ERROR
PMERR_INV_IN_AREA
PMERR_PATH_LIMIT_EXCEEDED
PMERR_INV_IN_PATH
PMERR_PATH_UNKNOWN
PMERR_HDC_BUSY
PMERR_DEV_FUNC_NOT_INSTALLED
PMERR_COORDINATE_OVERFLOW
PMERR_BITMAP_NOT_SELECTED

PMERR_INV_LENGTH_OR_COUNT
PMERR_INV_RECT
PMERR_INV_HDC
PMERR_INV_COORD_SPACE
PMERR_INV_COLOR_DATA
PMERR_INV_COLOR_INDEX
PMERR_INV_PICK_APERTURE_POSN.

GreDrawLinesInPath

```
(LONG) GreDrawLinesInPath (hdc, pPath, pLine, cLines, pInstance, lFunction)
```

Draws a sequence of one or more straight lines from the sequence of linked structures addressed by pLine.

The structures can be a mixture of LINE, CURVE, and FILLETSHARP structures; these have similar forms and the second field, bType, identifies the type of the structure. Starting at the structure addressed by pLine, the handling routine examines the bType field of each structure in turn; if bType is LINE_IDENTIFIER, the handling routine draws the line, otherwise it uses the value of the npcvNext field to skip to the next structure. This process continues until the handling routine has drawn the number of lines specified by cLines.

Before drawing a line, the handling routine must check the CURVE_DO_FIRST_PEL flag to determine whether it should draw the first pel of the line.

When the line passes between two pel positions, the driver should round down to the nearest pel for values of 0.5 or less.

The call to GreDrawLinesInPath in the presentation driver is made by the graphics engine. Coordinates are passed as screen coordinates (device coordinates + DC origin) and the lines are already completely clipped.

Support

This function must be supported by all presentation drivers.

The sample driver handles this call by bit-map simulation. The call parameters are passed unchanged to the display driver's dispatch table.

Stack Frame

(HDC)	hdc	(device-context handle)
(PPATH)	pPath	(pointer to a path structure, this parameter is ignored by the presentation driver)
(PCURVE)	pLine	(pointer to the first of a series of linked structures, see below)
(ULONG)	cLines	(count of LINE structures to draw)
(ULONG)	pInstance	(pointer to instance data)
(ULONG)	lFunction	(High-order word = Flags, Low-order word = NGreDrawLinesInPath)

pLine A pointer to the first structure in the path definition containing the lines that have to be drawn.

The LINE structure, shown below, is an example of the more general CURVE structure. These two structures, and the FILLETSHARP structure, are defined in PMDDI.H, they all take the same general form and are distinguished by the value of the bType field.

The LINE structure is defined as:

bIdent Identifier; this value has no significance for the driver.

bType Structure type. The only significant value is LINE_IDENTIFIER, which indicates that this structure is a LINE structure. If any other value is detected, the handling routine should skip to the structure addressed by npcvNext.

usStyle Line style. See "Line and Curve Styling" on page 12-2.

fs Flags. The only significant flag is:

CURVE_DO_FIRST_PEL When set, the handling routine must draw the first pel in the line. See also "First and Last Pel Considerations" on page 12-3.

npcvNext Near pointer to next structure in the sequence.

npcvPrev Near pointer to previous structure.

Reserved1[2] Reserved parameter.

ptfxA	Start of clipping limit, in the major direction of the line. (See Note below.)
ptfxC	End of clipping limit. (See Note below.)
PtsA	Start point of unclipped line, no pels are drawn before this point. (See Note below.)
PtsC	End point of unclipped line, no pels are drawn after this point. (See Note below.)
lRslope	This is ignored by the presentation driver.
Reserved2[4]	Reserved parameter.

Note: Start and end points are inclusive.

Return Codes

On completion, the handling routine must return an integer (cHits) indicating, where appropriate, whether correlation hits were detected:

GPI_OK	Successful.
GPI_HITS	Successful with correlate hit (returned by display drivers when the correlate flag is on and a hit is detected).
GPI_ERROR	Error.

When an error is detected, the handling routine must call WinSetErrorInfo to post the condition. Error codes for conditions that the handling routine is expected to check include:

PMERR_INV_IN_AREA
PMERR_INV_IN_PATH
PMERR_HDC_BUSY
PMERR_DEV_FUNC_NOT_INSTALLED
PMERR_COORDINATE_OVERFLOW
PMERR_BITMAP_NOT_SELECTED
PMERR_INV_LENGTH_OR_COUNT
PMERR_INV_HDC
PMERR_INV_COORD_SPACE
PMERR_INV_COLOR_DATA
PMERR_INV_COLOR_INDEX
PMERR_INV_PICK_APERTURE_POSN.

GrePolyShortLine

```
(LONG) GrePolyShortLine (hdc, pSline, pInstance, lFunction)
```

Draws a series of short lines. The current (x,y) position is not changed by this function.

A PolyShortLine is a linked list of short lines. The function renders each short line in the list until a null pslhNext is encountered.

Coordinates are passed as screen coordinates, and are already completely clipped.

Support

This function must be supported by all presentation drivers except those that hook GrePolyLine and all of the Arc functions that are simulated by handling routines in the graphics engine. (All calls to GrePolyShortLine come from either GrePolyLine or the Arc functions.)

The sample driver handles this call by bit-map simulation. The call parameters are passed unchanged to the display driver's dispatch table.

Stack Frame

(HDC)	hdc	(device-context handle)
(PSHORTLINE))	pSline	(pointer to SHORTLINE structure, see below)
(ULONG)	pInstance	(pointer to instance data)
(ULONG)	lFunction	(High-order word = Flags, Low-order word = NGrePolyShortline)

pSline The short lines are defined in a list of linked SHORTLINE structures:

slh SHORTLINEHEADER structure:

- **ulStyle** line style.
- **ptsStart** (x,y) position of start (the start position is included in the line).
- **ptsStop** (x,y) position of end (the end position is not included in the line).
- **sxLeft** left edge of bounding rectangle.
- **sxRight** right edge of bounding rectangle. The boundaries of the rectangle are inclusive at the start points of the lines and exclusive at the stop points, regardless of the direction.
- **pslhNext** pointer to next short line.
- **pslhPrev** pointer to previous short line.

This structure is a discrete representation of a curve that starts at point (x0, y0) and ends at point (x1, y1). For each of the (y1-y0+1) rows, there is exactly one x value contained in the x array – (the final point in the series is not drawn).

ax Steps. This is an array of x coordinates in absolute values. There is one coordinate value for each short line.

Return Codes

On completion, the handling routine must return an integer (cHits) indicating, where appropriate, whether correlation hits were detected:

GPI_OK Successful.

GPI_HITS Successful with correlate hit (returned by display drivers when the correlate flag is on and a hit is detected).

GPI_ERROR Error.

When an error is detected, the handling routine must call WinSetErrorInfo to post the condition. Error codes for conditions that the handling routine is expected to check include:

PMERR_INV_IN_AREA
PMERR_INV_IN_PATH
PMERR_HDC_BUSY
PMERR_DEV_FUNC_NOT_INSTALLED
PMERR_COORDINATE_OVERFLOW

PMERR_BITMAP_NOT_SELECTED
PMERR_INV_LENGTH_OR_COUNT
PMERR_INV_HDC
PMERR_INV_COORD_SPACE
PMERR_INV_COLOR_DATA
PMERR_INV_COLOR_INDEX
PMERR_INV_PICK_APERTURE_POSN.

GrePolyScanLine

```
(LONG) GrePolyScanLine (hdc, pScanData, pInstance, lFunction)
```

Fills an area lying between PolyShortLine pairs using the current pattern attribute.

Coordinates are passed as unclipped screen coordinates. Filling is inclusive at the left boundaries and exclusive at the right boundaries.

The scan lines are ordered from bottom to top and from left to right.

Support

This function must be supported by all presentation drivers except those that hook the GreEndArea and GreFillPath functions. (All calls to GrePolyScanLine come from either the GreEndArea or GreFillPath handling routine in the graphics engine.)

The sample driver handles this call by bit-map simulation. The call parameters are passed unchanged to the driver's bit-map simulation dispatch table.

Stack Frame

(HDC)	hdc	(device-context handle)
(PSCANDATA)	pScanData	(pointer ScanData structure)
(ULONG)	pInstance	(pointer to instance data)
(ULONG)	lFunction	(High-order word = flags, Low-order word = NGrePolyScanLine)

pScanData SCANDATA structure:

- **psiFirstLeft** Pointer to the left-hand end of the first PolyShortLine.
- **psiLastLeft** Pointer the left-hand end of the last PolyShortLine.
- **psiFirstRight** Pointer to right-hand edge of first PolyShortLine.
- **psiLastRight** Pointer to right-hand edge of last PolyShortLine.
- **c** Number of scan lines.
- **rclBound** Pointer to RECTL structure defining the bounding rectangle.

PolyShortLine The short lines are defined in a list of linked SHORTLINE structures:

- **slh** SHORTLINEHEADER structure:
 - **ulStyle** line style.
 - **ptsStart** (x,y) position of start.
 - **ptsStop** (x,y) position of end.
 - **sxLeft** left edge of bounding rectangle.
 - **sxRight** right edge of bounding rectangle.
 - **psihNext** pointer to next short line.
 - **psihPrev** pointer to previous short line.

 This structure is a discrete representation of a curve that starts at point (x0, y0) and ends at point (x1, y1). For each of the (y1-y0+1) rows, there is exactly one x value contained in the x array − (the final point in the series is not drawn).
- **ax** Array of x values, as absolute coordinates.

Return Codes

On completion, the handling routine must return an integer (cHits) indicating, where appropriate, whether correlation hits were detected:

- **GPI_OK** Successful.
- **GPI_HITS** Successful with correlate hit (returned by display drivers when the correlate flag is on and a hit is detected).
- **GPI_ERROR** Error.

When an error is detected, the handling routine must call WinSetErrorInfo to post the condition. Error codes for conditions that the handling routine is expected to check include:

PMERR_INV_IN_AREA
PMERR_INV_IN_PATH
PMERR_HDC_BUSY
PMERR_DEV_FUNC_NOT_INSTALLED
PMERR_COORDINATE_OVERFLOW
PMERR_BITMAP_NOT_SELECTED
PMERR_INV_LENGTH_OR_COUNT
PMERR_INV_RECT
PMERR_INV_COORDINATE
PMERR_INV_HDC
PMERR_INV_REGION_CONTROL
PMERR_INV_COORD_SPACE
PMERR_INV_COLOR_DATA
PMERR_INV_COLOR_INDEX
PMERR_INV_PICK_APERTURE_POSN.

Remarks

The handling routine can assume that:

- The two PolyShortLines do not cross
- Both PolyShortLines have the same height.

Text Functions

- GreCharString
- GreCharStringPos
- GreQueryTextBox
- GreQueryCharPositions
- GreQueryWidthTable.

```
(LONG) GreCharString (hdc, cChars, pchString, pInstance, lFunction)
```

Draws a character string starting at the current (x,y) position. Upon completion, the current (x,y) position is the start point for the character cell immediately after the last character in the string.

Support

GreCharString must be supported by the presentation driver. The handling routine must provide full support for drawing characters from an image font in CM_MODE1 when the character direction is CHDIRN_LEFTRIGHT (see "Character Attributes" on page 12-56). For outline characters or characters in any other mode or direction, the handling routine can dispatch the call to the graphics engine at the address given for this call in the default dispatch table.

The sample presentation driver handles this call by bit-map simulation. The call parameters are passed unchanged to the display driver's dispatch table.

Stack Frame

(HDC)	hdc	(device-context handle)
(LONG)	cChars	(number of characters in string)
(PCH)	pchString	(pointer to character string)
(ULONG)	pInstance	(pointer to instance data)
(ULONG)	lFunction	(High-order word = Flags, Low-order word = NGreCharString)

Return Codes

On completion, the handling routine must return a LONG value (cHits) indicating, where appropriate, whether correlation hits were detected:

GPI_OK Successful.
GPI_HITS Successful with correlate hit (returned by display drivers when the correlate flag is on and a hit is detected).
GPI_ERROR Error.

When an error is detected, the handling routine must call WinSetErrorInfo to post the condition. Error codes for conditions that the handling routine is expected to check include:

PMERR_ALREADY_IN_PATH
PMERR_BASE_ERROR
PMERR_BITMAP_NOT_SELECTED
PMERR_COORDINATE_OVERFLOW
PMERR_DEV_FUNC_NOT_INSTALLED
PMERR_EXCEEDS_MAX_SEG_LENGTH
PMERR_FONT_AND_MODE_MISMATCH
PMERR_HDC_BUSY
PMERR_HRGN_BUSY
PMERR_HUGE_FONTS_NOT_SUPPORTED
PMERR_INSUFFICIENT_MEMORY
PMERR_INV_BACKGROUND_COL_ATTR
PMERR_INV_BACKGROUND_MIX_ATTR
PMERR_INV_CHAR_ANGLE_ATTR
PMERR_INV_CHAR_DIRECTION_ATTR
PMERR_INV_CHAR_MODE_ATTR
PMERR_INV_CHAR_POS_OPTIONS
PMERR_INV_CHAR_SET_ATTR
PMERR_INV_CHAR_SHEAR_ATTR
PMERR_INV_CLIP_PATH_OPTIONS
PMERR_INV_CODEPAGE
PMERR_INV_COLOR_ATTR
PMERR_INV_COLOR_DATA
PMERR_INV_COLOR_INDEX
PMERR_INV_COORD_SPACE

PMERR_INV_COORDINATE
PMERR_INV_DC_TYPE
PMERR_INV_END_PATH_OPTIONS
PMERR_INV_FILL_PATH_OPTIONS
PMERR_INV_GEOM_LINE_WIDTH_ATTR
PMERR_INV_HDC
PMERR_INV_HRGN
PMERR_INV_ID
PMERR_INV_IN_AREA
PMERR_INV_IN_PATH
PMERR_INV_LENGTH_OR_COUNT
PMERR_INV_LINE_END_ATTR
PMERR_INV_LINE_JOIN_ATTR
PMERR_INV_LINE_TYPE_ATTR
PMERR_INV_LINE_WIDTH_ATTR
PMERR_INV_MARKER_SET_ATTR
PMERR_INV_MARKER_SYMBOL_ATTR
PMERR_INV_MATRIX_ELEMENT
PMERR_INV_MIX_ATTR
PMERR_INV_PATH_ID
PMERR_INV_PATTERN_REF_PT_ATTR
PMERR_INV_PATTERN_SET_ATTR
PMERR_INV_PATTERN_SET_FONT
PMERR_INV_PICK_APERTURE_POSN
PMERR_INV_PRIMITIVE_TYPE
PMERR_INV_RECT
PMERR_INV_REGION_CONTROL
PMERR_INV_SETID
PMERR_INV_TRANSFORM_TYPE
PMERR_NOT_IN_PATH
PMERR_PATH_LIMIT_EXCEEDED
PMERR_PATH_UNKNOWN
PMERR_REGION_IS_CLIP_REGION
PMERR_UNSUPPORTED_ATTR
PMERR_UNSUPPORTED_ATTR_VALUE.

```
(LONG) GreCharStringPos (hdc, pptStart, prcRect, flOptions, cChars,
                         pchString, padx, pAttrs, pInstance, lFunction)
```

Draws a character string. The string can be drawn from the current (x,y) position, or from a position specified.

Support

GreCharStringPos must be supported by the presentation driver. The handling routine must provide full support for drawing characters from an image font in CM_MODE1 when the character direction is CHDIRN_LEFTRIGHT (see "Character Attributes" on page 12-56). For outline characters or characters in any other mode or direction, the handling routine can dispatch the call to the graphics engine at the address given for this call in the default dispatch table.

The sample presentation driver handles this call by bit-map simulation. The call parameters are passed unchanged to the display driver's dispatch table.

Stack Frame

(HDC)	hdc	(device-context handle)
(PPOINTL)	pptStart	(pointer to (x,y) coordinates of start position)
(PRECTL)	prcRect	(pointer to an opaque or clip rectangle, see below)
(ULONG)	flOptions	(flags, see below)
(LONG)	cChars	(number of characters in string)
(PCH)	pchString	(pointer to character string)
(PLONG)	padx	(pointer to increment array, see below)
(PCSP_INFO)	pAttrs	(pointer to attributes structure, see below)
(ULONG)	pInstance	(pointer to instance data)
(ULONG)	lFunction	(High-order word = Flags, Low-order word = NGreCharStringPos)

prcRect The clipping rectangle pointed to by this parameter is defined as a RECTL structure:

xLeft	Minimum x coordinate of rectangle
yBottom	Minimum y coordinate
xRight	Maximum x coordinate of rectangle
yTop	Maximum y coordinate.

This rectangle, which is in world coordinates, is used as the clipping rectangle, or as the background, for the string, or both, depending on the value of flOptions. When the CHS_OPAQUE flag is set, normal background mix attributes are ignored, and the rectangle is drawn using overpaint and the character background color attribute.

When the CHS_OPAQUE is not set, the background is drawn, using the normal method. When neither CHS_OPAQUE nor CHS_CLIP are specified, this parameter is ignored.

Points on the boundary of this rectangle are considered to be inside the rectangle.

flOptions The following flags can be used in combination:

CHS_OPAQUE	Background of characters is defined by the rectangle prcRect. The rectangle is to be shaded (with background color and overpaint) before drawing.
CHS_VECTOR	Increment vector supplied (padx). If zero, padx is ignored.
CHS_LEAVEPOS	Leave current position at the start of string.
CHS_CLIP	Clip string to rectangle.
CHS_START_XY	Start position of the string. When set the handling routine must draw the string from the position indicated by pptStart. If this flag is not set the current position is used.

CHS_ATTR_INFO Attributes to be used. When this flag is set, pAttrs indicates the foreground and background colors. Current attributes are unchanged. If the flag is not set, the string is drawn using the current character attributes. See "Character Attributes" on page 12-56.

padx Pointer to an array of LONG integers; one element for each character in the string. When CHS_VECTOR is set, this array is used to set the spacing between characters. Each element is the distance, in world coordinates, from the bottom-left corner of the corresponding character in the string to the bottom-left corner of the next. The distance is measured along the baseline for left-to-right and right-to-left character directions, and along the shear line for top-to-bottom and bottom-to-top character directions. The final element is used reposition the current position, when necessary.

pAttrs Pointer to a CSP_INFO structure. This structure contains the attributes to be used to draw the string when the CHS_ATTR_INFO flag is set. These do not alter the current character attributes, see "Character Attributes" on page 12-56. The CSP_INFO structure is defined as:

cSize	Number of bytes in structure.
lColor	Use foreground color.
lBackColor	Use background color.

Return Codes

On completion, the handling routine must return a LONG value (cHits) indicating, where appropriate, whether correlation hits were detected:

GPI_OK	Successful.
GPI_HITS	Successful with correlate hit (returned by display drivers when the correlate flag is on and a hit is detected).
GPI_ERROR	Error.

When an error is detected, the handling routine must call WinSetErrorInfo to post the condition. Error codes for conditions that the handling routine is expected to check include:

PMERR_BASE_ERROR
PMERR_ALREADY_IN_PATH
PMERR_INV_IN_AREA
PMERR_INV_PATH_ID
PMERR_INV_END_PATH_OPTIONS
PMERR_NOT_IN_PATH
PMERR_PATH_LIMIT_EXCEEDED
PMERR_INV_FILL_PATH_OPTIONS
PMERR_INV_IN_PATH
PMERR_PATH_UNKNOWN
PMERR_INV_CLIP_PATH_OPTIONS
PMERR_HDC_BUSY
PMERR_DEV_FUNC_NOT_INSTALLED
PMERR_COORDINATE_OVERFLOW
PMERR_BITMAP_NOT_SELECTED
PMERR_INV_MATRIX_ELEMENT
PMERR_INV_PRIMITIVE_TYPE
PMERR_INV_SETID
PMERR_INV_CHAR_SET_ATTR
PMERR_INV_MARKER_SET_ATTR
PMERR_INV_PATTERN_SET_ATTR
PMERR_INV_MARKER_SYMBOL_ATTR
PMERR_UNSUPPORTED_ATTR
PMERR_INV_MIX_ATTR
PMERR_INV_BACKGROUND_MIX_ATTR
PMERR_INV_PATTERN_REF_PT_ATTR
PMERR_INV_LINE_WIDTH_ATTR
PMERR_INV_GEOM_LINE_WIDTH_ATTR

PMERR_INV_LINE_TYPE_ATTR
PMERR_INV_LINE_JOIN_ATTR
PMERR_INV_LINE_END_ATTR
PMERR_INV_CHAR_MODE_ATTR
PMERR_INV_CHAR_SHEAR_ATTR
PMERR_INV_CHAR_DIRECTION_ATTR
PMERR_UNSUPPORTED_ATTR_VALUE
PMERR_INSUFFICIENT_MEMORY
PMERR_INV_LENGTH_OR_COUNT
PMERR_INV_CODEPAGE
PMERR_EXCEEDS_MAX_SEG_LENGTH
PMERR_INV_RECT
PMERR_INV_COORDINATE
PMERR_INV_HDC
PMERR_INV_HRGN
PMERR_HRGN_BUSY
PMERR_INV_CHAR_ANGLE_ATTR
PMERR_INV_TRANSFORM_TYPE
PMERR_REGION_IS_CLIP_REGION
PMERR_INV_REGION_CONTROL
PMERR_INV_DC_TYPE
PMERR_INV_CHAR_POS_OPTIONS
PMERR_FONT_AND_MODE_MISMATCH
PMERR_INV_COORD_SPACE
PMERR_INV_ID
PMERR_INV_COLOR_DATA
PMERR_INV_COLOR_INDEX
PMERR_INV_PICK_APERTURE_POSN
PMERR_INV_PATTERN_SET_FONT
PMERR_HUGE_FONTS_NOT_SUPPORTED
PMERR_INV_COLOR_ATTR
PMERR_INV_BACKGROUND_COL_ATTR.

GreQueryTextBox

```
(BOOL) GreQueryTextBox (hdc, cChars, pchString, cptPosition, paptPosition, pInstance, lFunction)
```

Processes a character string as if it were being drawn. The handling routine stores the coordinates of the current text box, relative to the current (x,y) position, as an array at the location indicated by paptPosition. The first four coordinate pairs identify the bounding parallelogram for the given character string. The fifth coordinate pair is the (x,y) position of the starting point for the next character position after the string; that is the current position value that would be set by an equivalent GpiCharStringAt call. The positions take account of current values for character spacing such as kerning and character space. The points on the borders of the character box are deemed to be inside the box.

When the character mode is CM_MODE2, this function is only valid if the character angle, and shear attributes are set to their default values, see "Character Attributes" on page 12-56.

Support

This function must be supported by the presentation driver.

The sample presentation driver handles this call by bit-map simulation. The call parameters are passed unchanged to the display driver's dispatch table.

Stack Frame

(HDC)	hdc	(device-context handle)
(LONG)	cChars	(number of bytes in string)
(PCH)	pchString	(pointer to character string)
(LONG)	cptPosition	(number of (x,y) pairs that the Position array can contain)
(PPOINTL)	paptPosition	(pointer to position array, see below)
(ULONG)	pInstance	(pointer to instance data)
(ULONG)	lFunction	(High-order word = Flags, Low-order word = NGreQueryTextBox)

paptPosition Pointer to the array in which the function returns the requested values. Upon completion, the array contains cptPosition sets of (x,y) coordinates in the following order:

TXTBOX_TOPLEFT	Top left corner.
TXTBOX_BOTTOMLEFT	Bottom left corner.
TXTBOX_TOPRIGHT	Top right corner.
TXTBOX_BOTTOMRIGHT	Bottom right corner.
TXTBOX_CONCAT	Start point of next character position.

Return Codes

On completion, the handling routine must return BOOLEAN (fSuccess):

TRUE	Successful.
FALSE	Error.

When an error is detected, the handling routine must call WinSetErrorInfo to post the condition. Error codes for conditions that the handling routine is expected to check include:

PMERR_HDC_BUSY
PMERR_DEV_FUNC_NOT_INSTALLED
PMERR_COORDINATE_OVERFLOW
PMERR_INV_MATRIX_ELEMENT
PMERR_INV_SETID
PMERR_INV_CHAR_MODE_ATTR
PMERR_INSUFFICIENT_MEMORY
PMERR_INV_LENGTH_OR_COUNT
PMERR_INV_CODEPAGE
PMERR_EXCEEDS_MAX_SEG_LENGTH
PMERR_INV_HDC

PMERR_INV_CHAR_ANGLE_ATTR
PMERR_INV_TRANSFORM_TYPE
PMERR_INV_COORD_SPACE.

GreQueryCharPositions

```
(BOOL) GreQueryCharPositions (hdc, pptStart, flOptions, cChars, pchString, padx, paptXY, pInstance, lFunction)
```

Stores, at the location addressed by paptXY, an array of world coordinates identifying the start points at which the device is to place each character of a given string.

This function is required for the management of device fonts, in CM_MODE2 only. When the presentation driver has no device fonts, the handling routine must post PMERR_DEV_FUNC_NOT_INSTALLED.

Support

This function must be supported by the presentation driver.

The sample presentation driver handles this call by bit-map simulation. The call parameters are passed unchanged to the display driver's dispatch table.

Stack Frame

(HDC)	hdc	(device-context handle)
(PPOINTL)	pptStart	(pointer to (x,y) coordinates of optional starting position)
(ULONG)	flOptions	(Flags, see below)
(LONG)	cChars	(number of bytes in string)
(PCH)	pchString	(pointer to character string)
(PLONG)	padx	(pointer to Increment array, see below)
(PPOINTL)	paptXY	(pointer to array where character positions are returned)
(ULONG)	pInstance	(pointer to instance data)
(ULONG)	lFunction	(High-order word = Flags, Low-order word = NGreQueryCharPositions)

flOptions The following flags can be set:

CHS_VECTOR If set, increment vector is present.
CHS_START_XY If set, starting position is present, otherwise pptStart is ignored.

padx Vector of increment values, with one element for each character in the string. After writing a character, the increment specifies the absolute distance, in world coordinates, to get to the starting point of the next character.

paptxy Pointer to an array of (cChars + 1) returned positions. The first value in the array is the initial current position, the last value is the current position on return.

Return Codes

On completion, the handling routine must return BOOLEAN (fSuccess):

TRUE Successful.
FALSE Error.

When an error is detected, the handling routine must call WinSetErrorInfo to post the condition. Error codes for conditions that the handling routine is expected to check include:

PMERR_HDC_BUSY
PMERR_DEV_FUNC_NOT_INSTALLED
PMERR_COORDINATE_OVERFLOW
PMERR_INV_MATRIX_ELEMENT
PMERR_INV_SETID
PMERR_INV_CHAR_MODE_ATTR
PMERR_INSUFFICIENT_MEMORY
PMERR_INV_LENGTH_OR_COUNT
PMERR_INV_CODEPAGE
PMERR_EXCEEDS_MAX_SEG_LENGTH
PMERR_INV_HDC
PMERR_INV_CHAR_ANGLE_ATTR
PMERR_INV_TRANSFORM_TYPE

PMERR_INV_CHAR_POS_OPTIONS
PMERR_INV_COORD_SPACE.

GreQueryWidthTable

```
(BOOL) GreQueryWidthTable (hdc, lFirstChar, cWidthTable, paWidthTable, pInstance, lFunction)
```

Stores, at the location addressed by paWidthTable, an array of world coordinates representing the width-table information of the currently selected font. The handling routine must use GreConvert (page 14-99) to transform the width-table information from device to world coordinates.

Support

This function must be supported by the presentation driver.

The sample presentation driver handles this call by bit-map simulation. The call parameters are passed unchanged to the display driver's dispatch table.

Stack Frame

(HDC)	hdc	(device-context handle)
(LONG)	lFirstChar	(codepoint of the initial character for which width-table information is required)
(LONG)	cWidthTable	(count of widths in the width-table data)
(PLONG)	paWidthTable	(pointer to buffer of width-table data)
(ULONG)	pInstance	(pointer to instance data)
(ULONG)	lFunction	(High-order word = Flags, Low-order word = NGreQueryWidthTable)

Return Codes

On completion, the handling routine must return BOOLEAN (fSuccess):

TRUE Successful.
FALSE Error.

When an error is detected, the handling routine must call WinSetErrorInfo to post the condition. Error codes for conditions that the handling routine is expected to check include:

PMERR_COORDINATE_OVERFLOW
PMERR_DEV_FUNC_NOT_INSTALLED
PMERR_EXCEEDS_MAX_SEG_LENGTH
PMERR_HDC_BUSY
PMERR_INSUFFICIENT_MEMORY
PMERR_INV_CHAR_MODE_ATTR
PMERR_INV_CODEPAGE
PMERR_INV_COORD_SPACE
PMERR_INV_FIRST_CHAR
PMERR_INV_HDC
PMERR_INV_LENGTH_OR_COUNT
PMERR_INV_MATRIX_ELEMENT
PMERR_INV_SETID
PMERR_INV_TRANSFORM_TYPE.

Marker Functions

- GrePolyMarker.

GrePolyMarker

```
(LONG) GrePolyMarker (hdc, paptPoint, cPoints, pInstance, lFunction)
```

Draws a sequence of one or more markers: the first marker is drawn at the current (x,y) position; subsequent markers are drawn at the specified (x,y) positions, which indicate the centers of the markers. Upon completion, the current (x,y) position is the center of the last marker.

Support

This function must be supported by the presentation driver.

The sample presentation driver handles this call by bit-map simulation. The call parameters are passed unchanged to the display driver's dispatch table.

Stack Frame

(HDC)	hdc	(device-context handle)
(PPOINTL)	paptPoint	(pointer to points array, see below)
(LONG)	cPoints	(number of markers to be drawn)
(ULONG)	pInstance	(pointer to instance data)
(ULONG)	lFunction	(High-order word = Flags, Low-order word = NGrePolyMarker)

paptPoint An array of (cPoints) (x,y) pairs that contain the (x,y) coordinates of the markers.

cPoints When this is passed as null, the handling routine takes no action, except to return successful.

Return Codes

On completion, the handling routine must return an integer (cHits) indicating, where appropriate, whether correlation hits were detected:

GPI_OK Successful.
GPI_HITS Successful with correlate hit (returned by display drivers when the correlate flag is on and a hit is detected).
GPI_ERROR Error.

When an error is detected, the handling routine must call WinSetErrorInfo to post the condition. Error codes for conditions that the handling routine is expected to check include:

PMERR_BASE_ERROR
PMERR_ALREADY_IN_PATH
PMERR_INV_IN_AREA
PMERR_INV_PATH_ID
PMERR_INV_END_PATH_OPTIONS
PMERR_NOT_IN_PATH
PMERR_PATH_LIMIT_EXCEEDED
PMERR_INV_FILL_PATH_OPTIONS
PMERR_INV_IN_PATH
PMERR_PATH_UNKNOWN
PMERR_INV_CLIP_PATH_OPTIONS
PMERR_HDC_BUSY
PMERR_DEV_FUNC_NOT_INSTALLED
PMERR_COORDINATE_OVERFLOW
PMERR_BITMAP_NOT_SELECTED
PMERR_INV_MATRIX_ELEMENT
PMERR_INV_PRIMITIVE_TYPE
PMERR_INV_SETID
PMERR_INV_CHAR_SET_ATTR
PMERR_INV_MARKER_SET_ATTR
PMERR_INV_PATTERN_SET_ATTR
PMERR_INV_MARKER_SYMBOL_ATTR
PMERR_UNSUPPORTED_ATTR
PMERR_INV_MIX_ATTR

PMERR_INV_BACKGROUND_MIX_ATTR
PMERR_INV_PATTERN_REF_PT_ATTR
PMERR_INV_LINE_WIDTH_ATTR
PMERR_INV_GEOM_LINE_WIDTH_ATTR
PMERR_INV_LINE_TYPE_ATTR
PMERR_INV_LINE_JOIN_ATTR
PMERR_INV_LINE_END_ATTR
PMERR_INV_CHAR_MODE_ATTR
PMERR_INV_CHAR_SHEAR_ATTR
PMERR_INV_CHAR_DIRECTION_ATTR
PMERR_UNSUPPORTED_ATTR_VALUE
PMERR_INSUFFICIENT_MEMORY
PMERR_INV_LENGTH_OR_COUNT
PMERR_MATRIX_OVERFLOW
PMERR_INV_CODEPAGE
PMERR_EXCEEDS_MAX_SEG_LENGTH
PMERR_INV_RECT
PMERR_INV_COORDINATE
PMERR_INV_HDC
PMERR_INV_HRGN
PMERR_HRGN_BUSY
PMERR_INV_CHAR_ANGLE_ATTR
PMERR_INV_TRANSFORM_TYPE
PMERR_REGION_IS_CLIP_REGION
PMERR_INV_REGION_CONTROL
PMERR_INV_DC_TYPE
PMERR_INV_CHAR_POS_OPTIONS
PMERR_FONT_AND_MODE_MISMATCH
PMERR_INV_COORD_SPACE
PMERR_INV_ID
PMERR_INV_COLOR_DATA
PMERR_INV_COLOR_INDEX
PMERR_INV_PICK_APERTURE_POSN
PMERR_INV_PATTERN_SET_FONT
PMERR_HUGE_FONTS_NOT_SUPPORTED
PMERR_INV_COLOR_ATTR
PMERR_INV_BACKGROUND_COL_ATTR.

Bit Map Functions

- GreDeviceCreateBitmap
- GreDeviceDeleteBitmap
- GreDeviceSelectBitmap
- GreBitblt
- GreGetPel
- GreSetPel
- GreImageData
- GreDrawBorder
- GreGetBitmapBits
- GreSetBitmapBits.

Note: Presentation drivers for hard-copy vector devices should return fail on all bit map operations.

Bit-Map File Formats

The same bit-map file format is used for bit maps, icons, and pointers. For details refer to the *Programming Reference* and *C/2 Bindings Reference*.

```
(ULONG) GreDeviceCreateBitmap (hdc, pInfoHd, flUsage, pBitmap, pInfo, pInstance, lFunction)
```

Creates a bit map and obtains its handle.

Support

This function must be supported by the presentation driver.

The sample driver handles this call by bit-map simulation. The call parameters are passed unchanged to the display driver's dispatch table. The returned bit map handle is stored in the printer driver's own DC instance data structure. See module F0BITMAP.C of the sample presentation driver.

Stack Frame

(HDC)	hdc	(device-context handle)
(PBITMAPINFOHEADER)	pInfoHd	(pointer to BITMAPINFOHEADER structure, defining the new bit map)
(ULONG)	flUsage	(additional information used when creating a new bit map, see below)
(PBYTE)	pBitmap	(pointer to bit-map initialization data)
(PBITMAPINFO)	pInfo	(pointer to BITMAPINFO structure, see below)
(ULONG)	pInstance	(pointer to instance data)
(ULONG)	lFunction	(High-order word = flags, Low-order word = NGreDeviceCreateBitmap)

flUsage The only valid flag is:

CBM_INIT When set, the pBitmap and paInfo parameters are used to initialize the newly created bit map. The driver can assume that enough data is passed to initialize the whole bit map.

Other flags are reserved and must be ignored by the handling routine.

pInfo BITMAPINFO structure:

cbFix Length of structure.
cx Bit-map width.
cy Bit-map height.
cPlanes Number of planes (1 if standard format).
cBitCount Number of bits per pel.
argbColor[] Color table, as an array of RGB colors:
bBlue
bGreen
bRed.

Return Codes

On completion, the handling routine must return the bit-map handle (hbm), or GPI_ERROR, if an error is detected.

When an error is detected, the handling routine must call WinSetErrorInfo to post the condition. Error codes for conditions that the handling routine is expected to check include:

PMERR_DEV_FUNC_NOT_INSTALLED
PMERR_INSUFFICIENT_MEMORY
PMERR_INV_LENGTH_OR_COUNT
PMERR_INV_HDC
PMERR_INV_INFO_TABLE
PMERR_INV_BITMAP_DIMENSION
PMERR_INV_SCAN_START.

GreDeviceCreateBitmap

Remarks

Bit-map size is limited by available memory, the maximum width and height are 64KB.

There are several standard bit-map formats that should normally be adhered to. These are:

Bitcount	Planes
1	1
4	1
8	1
24	1

All presentation drivers must be able to create and use all of the standard formats but drivers for hard-copy devices may lose the color information. Drivers for display devices must maintain color information down to a minimum of 4 bits per pel.

The DC handle supplied to this function must never be null, bit maps always belong to some device. The bit map is created on the device specified. The bit map can be selected to a different device later as the graphics engine can handle transfer of bits from one device to the other.

When the value specified for cPlanes or cBitcount is incompatible with the physical device specified by the DC handle, an error is raised.

```
(BOOL) GreDeviceDeleteBitmap (hdc, hbm, pReturns, flOptions, pInstance, lFunction)
```

Destroys a bit map.

Support

This function must be supported by the presentation driver.

The sample driver handles this call by bit-map simulation. Before passing the call to the display driver's dispatch table, the handling routine must check the validity of the bit-map handle. Before returning, the printer driver deletes the bit-map handle in its own instance data structure. See module F0BITMAP.C of the sample presentation driver.

Stack Frame

Note: The handling routine must not use the values passed on the stack in the locations reserved for hdc and pInstance. These locations contain undefined data.

(HDC)	reserved	(see Note above)
(ULONG)	hbm	(handle of bit map to be destroyed)
(PDELETERETURN)	pReturns	(Pointer to returned bit-map parameters)
(ULONG)	flOptions	(additional information, used by the engine when creating, or deleting, a bit map, see below)
(ULONG)	reserved	(see Note above)
(ULONG)	lFunction	(High-order word = flags, Low-order word = NGreDeviceDeleteBitmap)

pReturns DELETERETURN structure:

lpInfo long pointer to BITMAPINFO structure
lpBits long pointer to bit map.

flOptions The only valid flag is:

CBM_INIT When set, bit-map parameters must be returned in pReturns. This means that before deleting the bit map, the handling routine must translate it into one of the standard formats. The driver must then allocate two blocks of memory, one for the bit map and one for the bit-map parameters and color translation table. The presentation driver can use any of the standard formats, but it must take into account the parameters originally specified in the GreDeviceCreateBitmap call. The handling routine should try to use the format that uses the least amount of memory, without loosing any bit-map information.

When this flag is not set, the bit-map data is not returned.

Other flags are reserved and should be ignored by the handling routine.

Return Codes

On completion, the handling routine must return BOOLEAN (fSuccess):

TRUE Successful.
FALSE Error.

When an error is detected, the handling routine must call WinSetErrorInfo to post the condition. Error codes for conditions that the handling routine is expected to check include:

PMERR_DEV_FUNC_NOT_INSTALLED
PMERR_INSUFFICIENT_MEMORY
PMERR_INV_LENGTH_OR_COUNT
PMERR_INV_HDC
PMERR_INV_INFO_TABLE
PMERR_INV_SCAN_START.

GreDeviceSelectBitmap

```
(BOOL) GreDeviceSelectBitmap (hdc, hbm, pInstance, lFunction)
```

Informs the presentation driver that a new bit map is being selected into the DC. See also "GreSelectBitmap" on page 15-20.

Support

This function must be supported by the presentation driver.

The sample driver handles this call by bit-map simulation. Before passing the call to the display driver, the handling routine must check the validity of the DC. The returned bit-map handle is stored in the printer's DC instance data structure. See module F0BITMAP.C of the sample presentation driver.

Stack Frame

(HDC)	hdc	(device-context handle)
(ULONG)	hbm	(device bit-map handle)
(ULONG)	pInstance	(pointer to instance data)
(ULONG)	lFunction	(High-order word = flags, Low-order word = NGreDeviceSelectBitmap)

Return Codes

On completion, the handling routine must return BOOLEAN (fSuccess):

TRUE Successful.
FALSE Error.

When an error is detected, the handling routine must call WinSetErrorInfo to post the condition. Error codes for conditions that the handling routine is expected to check include:

PMERR_DEV_FUNC_NOT_INSTALLED
PMERR_INV_HDC.

```
(LONG) GreBitblt (hdc, hdcSrc, cPoints, paptPoints, lRop, flOptions, pBattrs, pInstance, lFunction)
```

Modifies bit-map data at a target rectangle in the current DC; the modification can be either copying a rectangle of data from a specified source DC to the target or performing a raster operation on the target. The device contexts can be memory DCs, with bit maps selected, or DCs belonging to devices that support raster operations.

When copying bits from a color bit map to a monochrome bit map or device, only those pels that are in the source background color are copied to the target as background color. All other pels are copied to the target as foreground color.

Copying is nondestructive. When the target and source rectangles are in the same DC, no information is lost from the source if the rectangles overlap.

When the target is expressed in world coordinates (that is, the BBO_TARGWORLD flag is set in flOptions), these must be transformed to device coordinates and the bits are transferred to an upright rectangle in device space, regardless of any rotational elements that may have been present in the transforms.

The attribute structure identified by the pBattrs parameter defines the bit-map foreground/background colors. If pBattrs is null, the handling routine should use the current foreground and background colors.

When the mix specified by lRop requires both source and pattern, a 3-way operation is done using the current pattern in the target DC. If pattern mixing is not required, a 2-way operation is done.

If any of the source data is unavailable, the handling routine should transfer those bits that are present, and return without error. This could occur when, for example, the source DC is a window on the screen that has been overlaid by another. In this instance, the handling routine must proceed by reading what is there.

Support

This function must be supported by the presentation driver. However, if the destination is larger than the source, the presentation driver can choose to pass this call to the graphics engine using the engine pointer for GreBitblt copied from the default dispatch table.

The sample driver handles this call by bit-map simulation. The handling routine checks that the DC type is valid and then passes the parameters unchanged to the display driver's dispatch table. See module F0BITMAP.C of the sample presentation driver.

Stack Frame

(HDC)	hdc	(device-context handle)
(HDC)	hdcSrc	(source device-context or bit-map handle)
(LONG)	cPoints	(number of (x,y) pairs in paptPoints, see below)
(PPOINTL)	paptPoints	(pointer to an array of (x,y) coordinate pairs, see below)
(LONG)	lRop	(raster operation code, see below)
(ULONG)	flOptions	(specifies treatment of eliminated lines and columns when compression is done, see below)
(PBITBLTATTRS)	pBattrs	(pointer to attributes, see below)
(ULONG)	pInstance	(pointer to instance data)
(ULONG)	lFunction	(High-order word = flags, Low-order word = NGreBitblt)

hdcSrc Handle to the source DC or bit map. When lRop does not require a source, hdcSrc is passed as null, the handling routine then copies the current pattern to the currently selected bit map, or device.

Note: An engine bit-map handle is translated to a driver's bit-map handle in a call to GetDriverInfo.

cPoints A count of the number of (x,y) pairs in the paptPoints array. The count can be 2, 3, 4:

cPoints = 2 The operation is a raster operation, as determined by lRop, on the destination rectangle.

cPoints = 3 The operation is a copy between two rectangles of the same size. (Only the bottom-left corner is given for the source rectangle.)

cPoints = 4 The operation is determined by comparing the sizes of the two rectangles:

Target<Source: Compress the source rectangle into the target rectangle. The flOptions flags determine how you should handle eliminated rows and columns.

Target = Source: Copy between equal rectangles.

Target>Source: Stretch the source rectangle into the target rectangle. (In this case the call can be passed to the GreBitblt routine in the graphics engine.)

paptPoints A long pointer to a block of (x,y) coordinate pairs that define the target and source rectangles. The coordinates, which may be passed as a pair of RECTL structures, define the bottom-left and top-right corners of the target and source rectangles (but see cPoints above):

```
(xTgtBL, yTgtBL), (xTgtTR, yTgtTR), (xSrcBL, ySrcBL), (xSrcTR, ySrcTR)
```

When the source rectangle is totally, or partially, outside the source bit map (or device), the operation is implementation-dependent for that area. (That means that you have to decide what to do.)

Note: When BBO_TARGWORLD is not set, the rectangles are noninclusive. That is, they include the left and lower boundaries, in device units, but not the top and right boundaries. Thus, when the bottom-left corner of a rectangle maps to the same device pel as the top-right corner, that rectangle is considered to be empty.

When BBO_TARGWORLD is set, the target rectangle is inclusive at all boundaries, the source is noninclusive.

lRop Raster-operation code. The low-order byte represents a mix value in the range 00H through FFH. Raster-operation-code values and the mix-bit table are defined in the *Programming Reference*.

The handling routine uses this to determine the operations to perform on pattern, source, and target to get the required mix.

In addition to the ROP values defined at the API, the presentation driver must support ROP_GRAY (000080CAH). This value is used to shade the text for menu items that are not currently selectable.

When ROP_GRAY is set, the handling routine overpaints the foreground pattern using the current pattern and the background pattern color (background pels for the pattern are not changed). For the PATSYM_HALFTONE pattern, this overpaints the background pattern color onto alternate pels, leaving those in between unchanged.

flOptions Option flags:

BBO_OR	Stretch and compress as necessary, ORing any eliminated rows and columns. Used for white on black.
BBO_AND	Stretch and compress as necessary, ANDing any eliminated rows and columns. Used for black on white.
BBO_IGNORE	Stretch and compress as necessary, ignoring any eliminated rows and columns. Used for color.
BBO_TARGWORLD	The target rectangle is defined in world coordinates in the target PS.

When this option is specified, the target rectangle is transformed to device coordinates. Where any shear or rotation has occurred, this must be converted to an upright rectangle that bounds the transformed figure. This is then used as the target for the operation. No inversion of the image takes place.

BLTMODE_SRC_BITMAP
hdcSrc is a bit-map handle. The bit map must not be currently selected into a device context. (If this flag is not set, hdcSrc is a DC handle.)

BLTMODE_ATTRS_PRES
If set, the pBattrs parameter is present. This option can be ORed with any of the options above.

Note: Flags 15 through 31 are not used by the system; they are reserved for use by the driver.

pBattrs This points to a BITBLTATTRS structure:

cSize	Size of this structure.
lColor	Foreground color of source.
lBackColor	Background color of source.

The color values are used in conversions between monochrome and color data. This is the only format conversion required. The conversions are:

- Output of a monochrome pattern to a color device.

 In this instance, the source pattern is converted to a color pattern. This is done using the colors provided in the BITBLTATTRS structure. If these colors are not provided, the handling routine should use the current area colors for the target DC, see "Area (Pattern) Attributes" on page 12-54. The bits are then transferred so that:

 - Source 1s become (target area) foreground color
 - Source 0s become (target area) background color.

- Transfer from a monochrome bit map to a color bit map or device.

 In this instance, the source bits are converted using the current image colors. These are the colors provided in the BITBLTATTRS structure. If these colors are not provided, the handling routine should use the current image colors for the target DC, see "Image Attributes" on page 12-60. The bits are then transferred so that:

 - Source 1s become (target image) foreground color
 - Source 0s become (target image) background color.

- Transfer from a color bit map to a monochrome bit map or device.

 In this instance, the source bit map is converted using the source and target image colors. The target colors are provided in the BITBLTATTRS structure. If these colors are not provided, the handling routine should use those in the image attributes bundle for the target DC, see "Image Attributes" on page 12-60. When the source is a device context, the source-image-background color is that from the source DC. When the source is a bit-map handle, the background color is taken from the BITBLTATTRS structure, if provided, or otherwise from the background-image-color of the target DC. The bits are then transferred so that:

 - Source pels that are the source-image-background color become target-image-background color.
 - All other pels become target-image-foreground color.

When lRop does not call for a pattern, the pattern set and pattern symbol are not used.

Neither the source nor the pattern is required when a bit map, or part of a bit map, is being cleared to a particular color.

When a pattern is required, dithering can be done for solid patterns in a color that is not available on the device. Color dithering is described on page 12-101.

Return Codes

On completion, the handling routine must return an integer (cHits), indicating, where appropriate, whether correlation hits have been detected:

GPI_OK	Successful.
GPI_HITS	Successful with correlate hit (returned by display drivers when the correlate flag is on and a hit is detected).
GPI_ERROR	Error.

When an error is detected, the handling routine must call WinSetErrorInfo to post the condition. Error codes for conditions that the handling routine is expected to check include:

PMERR_BASE_ERROR
PMERR_INV_IN_AREA
PMERR_INV_IN_PATH
PMERR_HDC_BUSY
PMERR_DEV_FUNC_NOT_INSTALLED
PMERR_COORDINATE_OVERFLOW
PMERR_BITMAP_NOT_SELECTED
PMERR_INV_PRIMITIVE_TYPE
PMERR_INV_SETID
PMERR_INV_CHAR_SET_ATTR
PMERR_INV_MARKER_SET_ATTR
PMERR_INV_PATTERN_SET_ATTR
PMERR_INV_MARKER_SYMBOL_ATTR
PMERR_UNSUPPORTED_ATTR
PMERR_INV_MIX_ATTR
PMERR_INV_BACKGROUND_MIX_ATTR
PMERR_INV_PATTERN_REF_PT_ATTR
PMERR_INV_LINE_WIDTH_ATTR
PMERR_INV_GEOM_LINE_WIDTH_ATTR
PMERR_INV_LINE_TYPE_ATTR
PMERR_INV_LINE_JOIN_ATTR
PMERR_INV_LINE_END_ATTR
PMERR_INV_CHAR_MODE_ATTR
PMERR_INV_CHAR_SHEAR_ATTR
PMERR_INV_CHAR_DIRECTION_ATTR
PMERR_UNSUPPORTED_ATTR_VALUE
PMERR_INSUFFICIENT_MEMORY
PMERR_INV_LENGTH_OR_COUNT
PMERR_INV_CODEPAGE
PMERR_EXCEEDS_MAX_SEG_LENGTH
PMERR_INV_RECT
PMERR_INV_COORDINATE
PMERR_INV_HDC
PMERR_INV_HRGN
PMERR_INV_DC_DATA
PMERR_INCOMPATIBLE_BITMAP
PMERR_INV_REGION_CONTROL
PMERR_INV_DRIVER_NAME
PMERR_INV_BITBLT_STYLE
PMERR_INV_BITBLT_MIX
PMERR_INV_HBITMAP
PMERR_INV_DC_TYPE
PMERR_INV_COORD_SPACE
PMERR_INV_ID
PMERR_BITMAP_IS_SELECTED

PMERR_HBITMAP_BUSY
PMERR_INV_USAGE_PARM
PMERR_INV_INFO_TABLE
PMERR_INV_BITMAP_DIMENSION
PMERR_INCORRECT_DC_TYPE
PMERR_REALIZE_NOT_SUPPORTED
PMERR_INV_COLOR_FORMAT
PMERR_INV_COLOR_DATA
PMERR_INV_COLOR_START_INDEX
PMERR_INV_COLOR_OPTIONS
PMERR_INV_COLOR_INDEX
PMERR_INV_SCAN_START
PMERR_INV_PICK_APERTURE_POSN
PMERR_INV_PATTERN_SET_FONT
PMERR_HUGE_FONTS_NOT_SUPPORTED
PMERR_INV_COLOR_ATTR
PMERR_INV_BACKGROUND_COL_ATTR.

GreGetPel

```
(LONG) GreGetPel (hdc, pptPel, pInstance, lFunction)
```

Returns the color of a pel at a position specified in world coordinates. The return value of this function is either the color index of the pel, or its RGB value, depending on the color mode of the device.

Support

This function must be supported by the presentation driver.

The sample driver handles this call by bit-map simulation. The call parameters are passed unchanged to the display driver's dispatch table.

Stack Frame

(HDC)	hdc	(device-context handle)
(PPOINTL)	pptPel	(pointer to the coordinate (x,y) pair structure)
(ULONG)	pInstance	(pointer to instance data)
(ULONG)	lFunction	(High-order word = flags, Low-order word = NGreGetPel)

Return Codes

The handling routine must return the color index value for the pel, or CLR_NOINDEX if there is no index corresponding to the color.

The handling routine must raise an error when the point is subject to any clipping.

When an error is detected, the handling routine must call WinSetErrorInfo to post the condition. Error codes for conditions that the handling routine is expected to check include:

PMERR_HDC_BUSY
PMERR_DEV_FUNC_NOT_INSTALLED
PMERR_COORDINATE_OVERFLOW
PMERR_BITMAP_NOT_SELECTED
PMERR_INV_LENGTH_OR_COUNT
PMERR_INV_RECT
PMERR_INV_COORDINATE
PMERR_INV_HDC
PMERR_INV_COORD_SPACE
PMERR_PEL_NOT_AVAILABLE
PMERR_PEL_IS_CLIPPED.

```
(LONG) GreSetPel (hdc, pptPel, pInstance, lFunction)
```

Sets a pel to the current line attribute, color, and mix. This function is subject to all usual clipping, no error is returned when the point is clipped.

Support

This function must be supported by the presentation driver.

The sample driver handles this call by bit-map simulation. The call parameters are passed unchanged to the display driver.

Stack Frame

(HDC)	hdc	(device-context handle)
(PPOINTL)	pptPel	(pointer to pel position, in world coordinates)
(ULONG)	pInstance	(pointer to instance data)
(ULONG)	lFunction	(High-order word = flags, Low-order word = NGreSetPel)

Return Codes

On completion, the handling routine must return a LONG integer (cHits) indicating, where appropriate, whether correlation hits were detected:

GPI_OK Successful.
GPI_HITS Successful with correlate hit (returned by the display driver when the correlate flag is on and a hit is detected).
GPI_ERROR Error.

When an error is detected, the handling routine must call WinSetErrorInfo to post the condition. Error codes for conditions that the handling routine is expected to check include:

PMERR_INV_IN_AREA
PMERR_INV_IN_PATH
PMERR_HDC_BUSY
PMERR_DEV_FUNC_NOT_INSTALLED
PMERR_COORDINATE_OVERFLOW
PMERR_BITMAP_NOT_SELECTED
PMERR_INV_LENGTH_OR_COUNT
PMERR_INV_RECT
PMERR_INV_COORDINATE
PMERR_INV_HDC
PMERR_INV_COORD_SPACE
PMERR_INV_COLOR_DATA
PMERR_INV_COLOR_INDEX
PMERR_INV_PICK_APERTURE_POSN.

GreImageData

```
(LONG) GreImageData (hdc, pData, cBits, offRow, pInstance, lFunction)
```

Draws a single row of image data with one bit per pel using the current image foreground and background color and mix attributes. Drawing starts at the current x-position and at a y-position defined as a row offset below the current y-position. Data is supplied as a series of bytes and a count of the bits to be drawn; the handling routine cannot assume that unused bits at the end of the stream are set to zero. Bits are drawn from left to right. The high-order bit of each byte is the leftmost image bit. 1 bits are foreground and 0 bits are background.

This function does not affect the current position.

Support

This function must be supported by the presentation driver.

The sample driver handles this call by bit-map simulation. The call parameters are passed unchanged to the display driver.

Stack Frame

(HDC)	hdc	(device-context handle)
(PBYTE)	pData	(pointer to data string)
(LONG)	cBits	(number of bits in row, the maximum is 2040)
(ULONG)	offRow	(row offset, see below)
(ULONG)	pInstance	(pointer to instance data)
(ULONG)	lFunction	(High-order word = Flags, Low-order word = NGreImageData)

offRow The row number relative to the current y-position. Zero is the current position, a value of 1 indicates one row below the current position.

Return Codes

On completion, the handling routine must return a LONG integer (cHits), indicating, where appropriate, whether correlation hits have been detected:

GPI_OK Successful.
GPI_HITS Successful with correlate hit (returned by display drivers when the correlate flag is on and a hit is detected).
GPI_ERROR Error.

When an error is detected, the handling routine must call WinSetErrorInfo to post the condition. Error codes for conditions that the handling routine is expected to check include:

PMERR_INV_IN_AREA
PMERR_INV_IN_PATH
PMERR_HDC_BUSY
PMERR_DEV_FUNC_NOT_INSTALLED
PMERR_COORDINATE_OVERFLOW
PMERR_BITMAP_NOT_SELECTED
PMERR_INV_LENGTH_OR_COUNT
PMERR_INV_RECT
PMERR_INV_COORDINATE
PMERR_INV_HDC
PMERR_INV_REGION_CONTROL
PMERR_INV_COORD_SPACE
PMERR_INV_COLOR_DATA
PMERR_INV_COLOR_INDEX
PMERR_INV_SCAN_START
PMERR_INV_PICK_APERTURE_POSN
PMERR_INV_IMAGE_DATA_LENGTH.

```
(BOOL) GreDrawBorder (hdc, prcFrame, cxBorder, cyBorder, clrBorder, clrInterior, flCmd, pInstance, lFunction)
```

A fast-frame function that draws an internal border in a rectangular frame. The interior can also be filled.

Support

This function must be hooked by all presentation drivers. Drivers for printer and plotter devices do nothing except return successful. Display drivers must return failure if fast-frame drawing is not supported.

The sample presentation driver handles this call by bit-map simulation. The call parameters are passed unchanged to the display driver.

Stack Frame

(HDC)	hdc	(device-context handle)
(PRECTL)	prcFrame	(pointer to RECTL structure defining the frame in screen coordinates)
(ULONG)	cxBorder	(thickness of vertical border, in device coordinates)
(ULONG)	cyBorder	(thickness of horizontal border, in device coordinates)
(LONG)	clrBorder	(color of border, in any valid format)
(LONG)	clrInterior	(color of interior, in any valid format)
(ULONG)	flCmd	(options, see below)
(ULONG)	pInstance	(pointer to instance data)
(ULONG)	lFunction	(High-order word = Flags, Low-order word = NGreDrawBorder)

prcFrame Pointer to a RECTL structure. Coordinates are passed in screen coordinates, they are inclusive at the bottom-left corner and exclusive at the top-right corner.

flCmd Valid values are:

DB_ROP An option flag that defines the mix to be used for the border and the interior. The mixes are mutually exclusive with each other but can be combined independently with DB_INTERIOR or DB_AREAATTRS. This option can have any of the following values:

- **DB_PATCOPY** This is the default value. Use ROP_PATCOPY (see "GreBitblt" on page 12-35). This is a copy of the pattern to the destination.
- **DB_PATINVERT** Exclusive-OR the current pattern and the destination (ROP_PATINVERT), current mix and color parameters are ignored.
- **DB_DESTINVERT** This inverts the destination (ROP_DESTINVERT).
- **DB_AREAMIXMODE** This maps the current area foreground-mix attribute to a Bitblt raster operation. The area background-mix mode is ignored.

DB_INTERIOR Fill the area defined by prcFrame, excluding the border defined by cxBorder and cyBorder.

DB_AREAATTRS When set, the pattern used for the border is that currently defined in the area attribute. The pattern used for the interior is that which would be obtained by calling GreSetAttributes with the area attribute background color being passed for the foreground color and the area attribute foreground color being passed for the background. (See "GreSetAttributes" on page 15-25.)

When this flag is not set (default), the border pattern is equivalent to using GreSetAttributes for the area attributes using clrBorder as foreground color and clrInterior as the background. The Interior pattern is equivalent to using GreSetAttributes for the area attributes, using clrInterior color as foreground color, and clrBorder as the background.

The handling routine should ignore the remaining flags (DB_STANDARD and DB_DLGBORDER), which are used by the frame manager.

Return Codes

On completion, the handling routine must return BOOLEAN (fSuccess):

TRUE Successful.
FALSE Error.

When an error is detected, the handling routine must call WinSetErrorInfo to post the condition. Error codes for conditions that the handling routine is expected to check include:

PMERR_BASE_ERROR
PMERR_INV_IN_AREA
PMERR_INV_IN_PATH
PMERR_HDC_BUSY
PMERR_DEV_FUNC_NOT_INSTALLED
PMERR_COORDINATE_OVERFLOW
PMERR_BITMAP_NOT_SELECTED
PMERR_INV_PRIMITIVE_TYPE
PMERR_INV_SETID
PMERR_INV_CHAR_SET_ATTR
PMERR_INV_MARKER_SET_ATTR
PMERR_INV_PATTERN_SET_ATTR
PMERR_INV_MARKER_SYMBOL_ATTR
PMERR_UNSUPPORTED_ATTR
PMERR_INV_MIX_ATTR
PMERR_INV_BACKGROUND_MIX_ATTR
PMERR_INV_PATTERN_REF_PT_ATTR
PMERR_INV_LINE_WIDTH_ATTR
PMERR_INV_GEOM_LINE_WIDTH_ATTR
PMERR_INV_LINE_TYPE_ATTR
PMERR_INV_LINE_JOIN_ATTR
PMERR_INV_LINE_END_ATTR
PMERR_INV_CHAR_MODE_ATTR
PMERR_INV_CHAR_SHEAR_ATTR
PMERR_INV_CHAR_DIRECTION_ATTR
PMERR_UNSUPPORTED_ATTR_VALUE
PMERR_INSUFFICIENT_MEMORY
PMERR_INV_LENGTH_OR_COUNT
PMERR_INV_CODEPAGE
PMERR_EXCEEDS_MAX_SEG_LENGTH
PMERR_INV_RECT
PMERR_INV_COORDINATE
PMERR_INV_HDC
PMERR_INV_HRGN
PMERR_INV_DC_DATA
PMERR_INCOMPATIBLE_BITMAP
PMERR_INV_REGION_CONTROL
PMERR_INV_DRIVER_NAME
PMERR_INV_BITBLT_STYLE
PMERR_INV_BITBLT_MIX
PMERR_INV_HBITMAP
PMERR_INV_DC_TYPE
PMERR_INV_COORD_SPACE
PMERR_INV_ID
PMERR_BITMAP_IS_SELECTED
PMERR_HBITMAP_BUSY
PMERR_INV_USAGE_PARM
PMERR_INV_INFO_TABLE
PMERR_INV_BITMAP_DIMENSION
PMERR_INCORRECT_DC_TYPE

PMERR_REALIZE_NOT_SUPPORTED
PMERR_INV_COLOR_FORMAT
PMERR_INV_COLOR_DATA
PMERR_INV_COLOR_START_INDEX
PMERR_INV_COLOR_OPTIONS
PMERR_INV_COLOR_INDEX
PMERR_INV_DRAW_BORDER_OPTION
PMERR_INV_SCAN_START
PMERR_INV_PICK_APERTURE_POSN
PMERR_INV_PATTERN_SET_FONT
PMERR_HUGE_FONTS_NOT_SUPPORTED
PMERR_INV_COLOR_ATTR
PMERR_INV_BACKGROUND_COL_ATTR.

Remarks

cxBorder and cyBorder can be passed as zero. When both are zero, the interior must still be drawn. When the x-borders and y-borders overlap, the border is drawn as a single rectangle with no interior.

This function is a Bitblt accelerator and has similar function, and limitations, to GreBitblt, GreDeviceSetAttributes, and GreSetAttributes.

See the *Programming Reference* for a description of the WinDrawBorder function.

GreGetBitmapBits

```
(LONG) GreGetBitmapBits (hdc, hbm, iScanStart, cScanCount, pAddress, pInfo, pInstance, lFunction)
```

Transfers bit-map data from a bit map to application storage.

The bit map can be specified by a bit-map handle or, if this is null, a DC handle, in which instance the device context must be a memory DC with a bit map currently selected.

Support

This function must be supported by the presentation driver.

The sample driver handles this call by bit-map simulation. Before the call is passed to the display driver, the printer driver checks that the DC and bit map are valid. See module F0BITMAP.C of the sample presentation driver. See also module U1BANDIN.C where this function is used to retrieve a bit map from the display driver.

Stack Frame

(HDC)	hdc	(device-context handle)
(HBITMAP)	hbm	(bit-map handle, if zero the DC bit map is used)
(LONG)	iScanStart	(scan-line number from where data transfer starts, zero is the first)
(LONG)	cScanCount	(number of scan lines to be transferred)
(PBYTE)	pAddress	(pointer to receiving area for bit-map data)
(PBITMAPINFO)	pInfo	(pointer to BITMAPINFO structure, see below)
(ULONG)	pInstance	(pointer to instance data)
(ULONG)	lFunction	(High-order word = flags, Low-order word = NGreGetBitmapBits)

pInfo Pointer to a BITMAPINFO structure:

cbFix	Length of structure.
cx	Bit-map width, this value is updated by the handling routine.
cy	Bit-map height, this value is updated by the handling routine.
cPlanes	Number of planes (1 if standard format), set before call is made.
cBitCount	Number of bits per pel, set before call is made.
argbColors	Color table, this value is updated by the handling routine.

Return Codes

On completion, the handling routine must return a LONG value (lLines) indicating the number of lines transferred, or GPI_ALTERROR if an error occurred.

When an error is detected, the handling routine must call WinSetErrorInfo to post the condition. Error codes for conditions that the handling routine is expected to check include:

PMERR_INV_IN_AREA
PMERR_INV_IN_PATH
PMERR_DEV_FUNC_NOT_INSTALLED
PMERR_BITMAP_NOT_SELECTED
PMERR_INV_LENGTH_OR_COUNT
PMERR_INV_HBITMAP
PMERR_INV_INFO_TABLE
PMERR_INCORRECT_DC_TYPE
PMERR_INV_SCAN_START.

Remarks

When the bit-map handle is null, the DC must be a memory DC with a bit map currently selected, otherwise the DC handle must be valid for that device.

The BITMAPINFO structure must be initialized with the values of cPlanes and cBitcount, for the format of data required. This must be one of the standard formats, or a device-specific format that matches the DC. On return, cx, cy, and argbColors are supplied by the system.

Conversion of the bit-map data is carried out if necessary.

pAddress must point to a storage area large enough to contain data for the requested number of scan lines. The amount of storage required for one scan line can be determined by calling GetBitmapParameters. It is:

```
((cBitcount * cx + 31)/32) * cPlanes * 4   bytes
```

GreSetBitmapBits

```
(LONG) GreSetBitmapBits (hdc, hbm, iScanStart, cScanCount, pAddress, pInfo, pInstance, lFunction)
```

Transfers bit-map data from application storage into the specified bit map or DC.

The bit map can be specified by its handle or, if this is null, a DC handle, in which instance, the device context must be a memory DC with a bit map currently selected. This function does not set bits, directly, into any other kind of device. When the format of the supplied bit map does not match that of the device, the handling routine must convert it, using the supplied BITMAPINFO structure. Only standard formats, and device formats that are compatible with the target device, are supported.

Support

This function must be supported by the presentation driver.

The sample driver handles this call by bit-map simulation. Before passing the call to the display driver, the handling routine checks that the DC and bit map are valid. See module F0BITMAP.C of the sample presentation driver.

Stack Frame

(HDC)	hdc	(device-context handle)
(HBITMAP)	hbm	(bit-map handle)
(LONG)	iScanStart	(scan-line number from where data transfer starts, zero is the first)
(LONG)	cScanCount	(number of scan lines to be transferred)
(PBYTE)	pAddress	(pointer to receiving area for bit-map data
(PBITMAPINFO)	pInfo	(pointer to BITMAPINFO structure, see below)
(ULONG)	pInstance	(pointer to instance data)
(ULONG)	lFunction	(High-order word = flags, Low-order word = NGreSetBitmapBits)

pInfo Pointer to a BITMAPINFO structure:

cbFix	Length of structure.
cx	Bit-map width.
cy	Bit-map height.
cPlanes	Number of planes (1 if standard format).
cBitCount	Number of bits per pel.
argbColors	Color table.

Return Codes

On completion, the handling routine must return a LONG value (lLines) indicating the number of lines transferred, or GPI_ALTERROR if an error occurred.

When an error is detected, the handling routine must call WinSetErrorInfo to post the condition. Error codes for conditions that the handling routine is expected to check include:

PMERR_INV_IN_AREA
PMERR_INV_IN_PATH
PMERR_DEV_FUNC_NOT_INSTALLED
PMERR_BITMAP_NOT_SELECTED
PMERR_INV_LENGTH_OR_COUNT
PMERR_INV_HBITMAP
PMERR_INV_INFO_TABLE
PMERR_INCORRECT_DC_TYPE
PMERR_INV_SCAN_START.

Remarks

When the bit-map handle is null, the DC must be a memory DC with a bit map currently selected. Otherwise the DC must be valid for that device.

When the format of the supplied bit map does not match that of the device, the handling routine must use the supplied BITMAPINFO structure to convert it.

Only the standard formats, or device-specific formats, are supported.

Attribute Functions

- GreDeviceGetAttributes
- GreGetPairKerningTable
- GreDeviceSetAttributes
- GreDeviceSetGlobalAttribute.

Attribute and Bundle Definitions

This section provides a general description of colors, mixes, and patterns. The attribute definitions, for each attribute bundle type, are also provided.

For area definitions, the area must be filled using the pattern that is current when GreBeginArea is called.

Colors

All colors are passed as 32-bit signed values. These are either indexes into the logical color table or representations of 24-bit red, green, and blue (RGB) values.

Some special attribute values can be passed to the graphics engine and returned by GreGetAttributes:

CLR_FALSE	All color planes, or bits, or both, are 0.
CLR_TRUE	All color planes, or bits, or both, are 1.
CLR_WHITE	This index is never loaded explicitly. It always produces white when the default color table is in force or when the index is set to RGB. With a realized color table and an index that is not RGB, CLR_WHITE produces the background color (CLR_BACKGROUND).
CLR_BLACK	This index is never loaded explicitly. It always produces black when the default color table is in force or when the index is set to RGB. With a realized color table and an index that is not RGB, CLR_BLACK produces the neutral color (CLR_NEUTRAL). See "The Logical Color Table" on page 12-92.

Note: CLR_DEFAULT is the default value at the API. It is a reserved value and is not passed to the presentation driver.

CLR_FALSE and CLR_TRUE provide useful operands for Bitblt logical operations.

Mix Modes

All values are passed to the graphics engine, which, in turn, passes them unchanged to the presentation driver.

Foreground-mix mode Valid values are:

FM_OR	OR
FM_OVERPAINT	Overpaint
FM_XOR	Exclusive-OR
FM_LEAVEALONE	Leave Alone (invisible)
FM_AND	AND
FM_SUBTRACT	(inverse source) AND destination
FM_MASKSRCNOT	Source AND (inverse destination)
FM_ZERO	All zeros
FM_NOTMERGESRC	Inverse (source OR destination)
FM_NOTXORSRC	Inverse (source exclusive-OR destination)
FM_INVERT	Inverse of destination
FM_MERGESRCNOT	Source OR (inverse destination)
FM_NOTCOPYSRC	Inverse of source
FM_MERGENOTSRC	(inverse source) OR destination
FM_NOTMASKSRC	Inverse of (source AND destination)
FM_ONE	All ones.

Note: FM_DEFAULT is the default value at the API. It is a reserved value and is not passed to the presentation driver.

The presentation driver must support FM_OR, FM_OVERPAINT, FM_XOR, and FM_LEAVEALONE. Other foreground mixes can be handled as FM_OVERPAINT.

Mixing, other than for FM_LEAVEALONE, or FM_OVERPAINT, is done on the physical color index. When an indexed color table has been realized, this corresponds to the logical color index. In other instances, the color resulting from a mix cannot be predicted.

Note: An exception to this rule is when the same object is drawn twice with FM_XOR and BM_LEAVEALONE, and no intermediate drawing in other mixes; the implementation must guarantee that this always results in the object being erased cleanly.

Other valid mixes can also be supported for some primitive types. When a valid mix is not supported, the default is FM_OVERPAINT. An error is only raised when the specified mix value is not one of those listed above.

Background Mix Mode Valid values are:

BM_ERROR	Error
BM_DEFAULT	Default
BM_OR	OR
BM_OVERPAINT	Overpaint
BM_XOR.	Exclusive-OR
BM_LEAVEALONE	Leave Alone (invisible).

The presentation driver must support BM_OVERPAINT and BM_LEAVEALONE. Other foreground mixes can be handled as BM_LEAVEALONE.

GreQueryDeviceCaps allows the application to determine the mixes supported by the device. See page 12-106.

Line (Pen) Attributes

The device line (pen) attributes are bundled in a DLINEBUNDLE structure:

cAttr	Size of the logical attribute bundle.
cDefs	Size of the LINEDEFS structure.
lbnd	LINEBUNDLE structure, this is the logical line bundle as seen by the application, see below.
ldef	LINEDEFS structure: **defType** Line definition, this is ignored by the presentation driver.

LINEBUNDLE Mask: This mask is used in calls to GreDeviceSetAttributes to identify fields in the LINEBUNDLE structure. Valid flags and the fields that they identify are:

Flag	Field
LBB_COLOR	lColor
LBB_MIX_MODE	usMixMode
LBB_WIDTH	fxWidth
LBB_GEOM_WIDTH	lGeomWidth
LBB_TYPE	usType
LBB_END	usEnd
LBB_JOIN	usJoin.

LINEBUNDLE Structure: The fields of a LINEBUNDLE structure are:

lColor Line foreground color.

usMixMode Line foreground-mix mode.

fxWidth Line-width multiplier. This value is expressed in fixed-point notation with a notional binary point between the second and third bytes, thus 1.0 is represented by 64KB. This multiplier is applied to the normal line width. Valid values are:

LINEWIDTH_DEFAULT	Default line width, this is the same as LINEWIDTH_NORMAL.
LINEWIDTH_NORMAL	Normal line width.
LINEWIDTH_THICK	Thick line width, this is double the normal width.

Typically, values equal or less than 1.0 should be treated as LINEWIDTH_NORMAL, and values greater than 1.0 as LINEWIDTH_THICK.

lGeomWidth Geometric line thickness, in world-coordinate space, specified as an integer value. This is used only by GreStrokePath or when MPATH_STROKE is specified for GreModifyPath.

A value of zero results in the thinnest line possible, regardless of the transform in force. Thick geometric lines are treated as polygons and are transformed accordingly.

usType Specifies the cosmetic line type. Valid values are:

LINETYPE_DOT	Dotted
LINETYPE_SHORTDASH	Short-dashed
LINETYPE_DASHDOT	Dash, dot
LINETYPE_DOUBLEDOT	Double-dotted
LINETYPE_LONGDASH	Long-dashed
LINETYPE_DASHDOUBLEDOT	Dash, double-dot
LINETYPE_SOLID	Solid
LINETYPE_INVISIBLE	Invisible
LINETYPE_ALTERNATE	Every alternate pel on.

Note: LINETYPE_DEFAULT is the default value at the API. It is a reserved value and is not passed to the presentation driver.

usEnd Valid values are:

LINEEND_FLAT	Flat
LINEEND_SQUARE	Square
LINEEND_ROUND	Round.

Note: LINEEND_DEFAULT is the default value at the API. It is a reserved value and is not passed to the presentation driver.

usJoin Valid values are:

LINEJOIN_BEVEL	Bevel
LINEJOIN_ROUND	Round
LINEJOIN_MITRE	Miter.

Note: LINEJOIN_DEFAULT is the default value at the API. It is a reserved value and is not passed to the presentation driver.

When lines join at a very acute angle and a mitered joint has been specified, then the length of the miter line could extend almost to infinity. To prevent this, when the ratio of miter length to geometric line width exceeds 10:1, a bevel joint is drawn. The miter length is the distance between the inner and outer intersection points.

Area (Pattern) Attributes

The device area (pattern) attributes are bundled in a DAREABUNDLE structure:

cAttr	Size of the logical area bundle.
cDefs	Size of the AREADEFS structure. This is zero when no device area bundle exists.
abnd	AREABUNDLE structure, see below.
adef	AREADEFS structure: **defSet** Area definition. This can be a text pattern, a predefined pattern, a bit map, a pointer to an engine font, a device-font handle, or a bit-map handle. **fFlags** The only valid flag is CDEF_GENERIC. **CodePage** Code page ID.

AREABUNDLE Mask: This mask is used in calls to GreDeviceSetAttributes to identify fields in the AREABUNDLE structure. Valid flags and the fields that they identify are:

Flag	Field
ABB_COLOR	lColor
ABB_BACK_COLOR	lBackColor
ABB_MIX_MODE	usMixMode
ABB_BACK_MIX_MODE	usBackMixMode
ABB_SET	usSet
ABB_SYMBOL	usSymbol
ABB_REF_POINT	ptlRefPoint.

AREABUNDLE Structure: The fields of an AREABUNDLE structure are:

lColor Area foreground color.

lBackColor Area background color.

usMixMode Area foreground-mix mode.

usBackMixMode Area background-mix mode.

usSet Local identifier (lcid) for a logical font or a bit map. Valid values are:

Zero Base pattern set.
Nonzero Local identifier for the logical font or bit map defined by the cdef.defSet field in the area-attributes bundle.

usSymbol Identity of the required pattern in the current pattern set, or logical font (this attribute is ignored when the pattern set is a bit map). If the value is outside the range of the logical font, the standard default pattern is used.

Values in the range 1 through 255 are valid. The defined values are:

PATSYM_ERROR	Error
PATSYM_DEFAULT	Default
PATSYM_DENSE1	Solid shading with decreasing intensity
PATSYM_DENSE2	..
PATSYM_DENSE3	..
PATSYM_DENSE4	..
PATSYM_DENSE5	..
PATSYM_DENSE6	..
PATSYM_DENSE7	..
PATSYM_DENSE8	..
PATSYM_VERT	Vertical lines
PATSYM_HORIZ	Horizontal lines
PATSYM_DIAG1	Diagonal lines 1, bottom left to top right
PATSYM_DIAG2	Diagonal lines 2, bottom left to top right
PATSYM_DIAG3	Diagonal lines 1, top left to bottom right
PATSYM_DIAG4	Diagonal lines 2, top left to bottom right
PATSYM_NOSHADE	No shading
PATSYM_SOLID	Solid shading
PATSYM_BLANK	Blank (same as PATSYM_NOSHADE)
PATSYM_HALFTONE	Every alternate pel on.

See the *Programming Reference* for definitions of these shading patterns.

Note: PATSYM_HALFTONE can be similar to PATSYM_DENSE4 and PATSYM_DENSE5 (solid patterns) but has a more stringent definition. It is useful for generating gray text.

ptlRefPoint Specifies the pattern origin for areas and thick lines. The pattern is mapped into the area to be filled by conceptually replicating the pattern definition in both the horizontal and vertical directions.

The pattern reference point is subject to all of the transforms. When an area is moved by changing a transform and redrawing, the fill pattern also appears to move so as to retain its position relative to the area boundaries. This allows part of a picture to be moved with a BitBlt operation, and the remainder to be drawn by changing the appropriate transform, with no discontinuity at the join.

The pattern reference point, which is specified in world coordinates, need not be inside the actual area to be filled. The pattern reference point is not subject to clipping, although the area to be filled is.

Character Attributes

The device character attributes are bundled in a DCHARBUNDLE structure:

cAttr	Size of the attribute bundle.
cDefs	Size of the CHARDEFS scructure.
cbnd	CHARBUNDLE structure, see below.
cdef	CHARDEFS structure: **defSet** Character set definition. If defSet is passed as zero, the driver must use the default device font (zero is passed only when the driver provides and manages its own default font). Otherwise the significance of defSet depends upon the state of the CDEF_GENERIC flag (see below): • If the flag is set, defSet is a long pointer to an engine font. • If the flag is not set, defSet is a device-font identifier defined by the driver. **fFlags** Valid flags are: **CDEF_GENERIC** Engine font (not device font) **CDEF_BOLD** Font must be emboldened **CDEF_ITALIC** Font must be italicized **CDEF_UNDERLINE** Font must be underlined **CDEF_STRIKEOUT** Font must be StrikeOut. **CodePage** Current code page. (The driver should ignore this field when the font is not a multi-codepage font that needs translating.) **charSpacing** Character spacing.

CHARBUNDLE Mask: This mask is used in calls to GreDeviceSetAttributes to identify fields in the CHARBUNDLE structure. Valid flags and the fields that they identify are:

Flag	Field
CBB_COLOR	lColor
CBB_BACK_COLOR	lBackColor
CBB_MIX_MODE	usMixMode
CBB_BACK_MIX_MODE	usBackMixMode
CBB_SET	usSet
CBB_MODE	usPrecision
CBB_BOX	sizfxCell
CBB_ANGLE	ptlAngle
CBB_SHEAR	ptlShear
CBB_DIRECTION	usDirection.

CHARBUNDLE Structure: The fields of a CHARBUNDLE structure are:

lColor Character foreground color.

lBackColor Character background color.

usMixMode Character foreground-mix mode.

usBackMixMode Character background-mix mode.

usSet Specifies a local identifier (lcid) for a logical font. If usSet is NULL, the current code page and character precision are used to resolve the selection of the base font. The code page (set by the function GreSetCodePage) identifies two base fonts, a outline font, and an image font. The value of usPrecision determines which of these is selected. Valid values for usSet are:

Zero Base-font.
Nonzero Local identifier for the logical font defined by the cdef.defSet field in the character-attributes bundle.

usPrecision Specifies the character mode. The value of usPrecision is used to select output quality from precision 1, the lowest, to precision 3. Presentation drivers would normally use precision 3 except when performance would be improved by using the specified precision. For example, the EGA and VGA drivers always use precision 3 when the current font is an outline font but switch to the specified precision for an image font.

Valid values for usPrecision are:

CM_MODE1 (Precision 1) The selected font can be either an image or an outline font. When an image font is used, the first character is positioned with its reference point at the current position and subsequent characters are positioned using the FONTMETRICS.

CM_MODE2 (Precision 2) The selected font can be either an image or outline font. When an image font is used, the first character is positioned with its reference point at the current position, subsequent characters are positioned using the FONTMETRICS, CBB_BOX, CBB_ANGLE, and CBB_SHEAR attributes.

This is done by constructing a transformation matrix that transforms the FONTMETRICS sXDeviceRes and sYDeviceRes values to the rotated, scaled and sheared character box attributes and which translates the reference point of the first character to the current position. The actual character positions are then determined by applying this transform to the character positions found using the FONTMETRICS.

Notes:

1. The character-box attribute is subjected to the rotation, scaling and shear defined by the current transformations as well as that defined by the other character attributes.
2. Shear attributes only affect the horizontal position of CM_MODE2 characters, and then only when the character direction is CHRDIRN_TOPBOTTOM or CHRDIRN_BOTTOMTOP.

CM_MODE3 (Precision 3) The selected font must be an outline font.

Note: CM_DEFAULT is the default value at the API. It is a reserved value and is not passed to the presentation driver.

For outline fonts, regardless of mode, all character attributes together with the FONTMETRICS are used for positioning, scaling, rotating and shearing the characters.

This is done by constructing a transformation matrix as described above for CM_MODE2. The actual character-vector-stroke coordinates are then determined by applying this transform to the character definition coordinates, suitably modified (for character positioning) by the FONTMETRICS.

Note: As with CM_MODE2, the character-box attribute is subjected to the rotation, scaling, and shear defined by the current transformations as well as that defined by the other character attributes.

The character reference point is defined as the intersection of the base line and the left-hand edge of the character. The base line is defined as an offset (pCellOffset) from the top of the character cell.

sizfxCell Specifies fixed-point numbers for the width and height of a character cell in world-coordinate space. This defines the background area for a character.

Each dimension is represented as a signed 4-byte integer, with a notional binary point between bits 16 and 15. Thus: +2.5 is represented by 00028000H and −2.5 is represented by FFFD8000H.

For CM_MODE1, the cell has no effect when characters are drawn from an image font. For CM_MODE2, the width determines the spacing of consecutive characters along the baseline.

Both width and height can be positive, negative, or zero. When either parameter is negative, the spacing occurs in the opposite direction to normal and each character is drawn reflected in CM_MODE3. Thus, for example, a negative height in the standard direction in mode 3 means that the characters are drawn upside down, and the string is drawn below the baseline (assuming no other transformations cause inversion).

A zero character width or height is also valid; here, the string of characters collapses into a line. If both are zero, the string is drawn as a single point in CM_MODE3.

ptlAngle Specifies integer values x and y for the coordinates of the end of a line starting at the origin (0,0); the base line for subsequent character strings is parallel to this line.

For CM_MODE1, the angle has no effect when characters are drawn.

For CM_MODE2, the angle is used to determine the position of each image character, but the orientation of characters within the character box is inherent in their definitions. The characters are positioned so that the lower left-hand corners of the character definitions are placed at the lower left-hand corners of the character boxes.

For CM_MODE3, the angle is observed accurately, and the character boxes are rotated to be normal to the character baseline. If the coordinate system is such that one x-axis unit is not physically equal to one y-axis unit, a rotated character appears to be sheared.

ptlShear Specifies integer values that identify the end coordinates of a line originating at 0,0; the vertical strokes in subsequent outline character strings are drawn parallel to the defined line. For CM_MODE1, the shear has no effect when image characters are drawn. For CM_MODE2, the shear affects the height of the character cell and hence the position of characters drawn with CDIRN_TOPBOTTOM or CDIRN_BOTTOMTOP.

The top of the character box remains parallel to the character baseline.

If hx = 0 and hy = 1 (the standard default), *upright* characters result. If hx and hy are both positive or both negative, the characters slope from bottom left to top right. If hx and hy are of opposite signs, the characters slope from top left to bottom right. No character inversion takes place as a result of shear alone. (Inversion can be done with the charCell attribute).

It is an error to specify a zero value for hy, because this would imply an infinite shear.

usDirection Valid values are:

CHDIRN_LEFTRIGHT	Left to right
CHDIRN_TOPBOTTOM	Top to bottom
CHDIRN_RIGHTLEFT	Right to left
CHDIRN_BOTTOMTOP	Bottom to top.

Note: CHDIRN_DEFAULT is the default value at the API. It is a reserved value and is not passed to the presentation driver.

If the specified direction is not valid, the presentation driver should use CHDIRN_LEFTRIGHT as the default.

Figure 12-2 shows how the character direction affects the origin of characters.

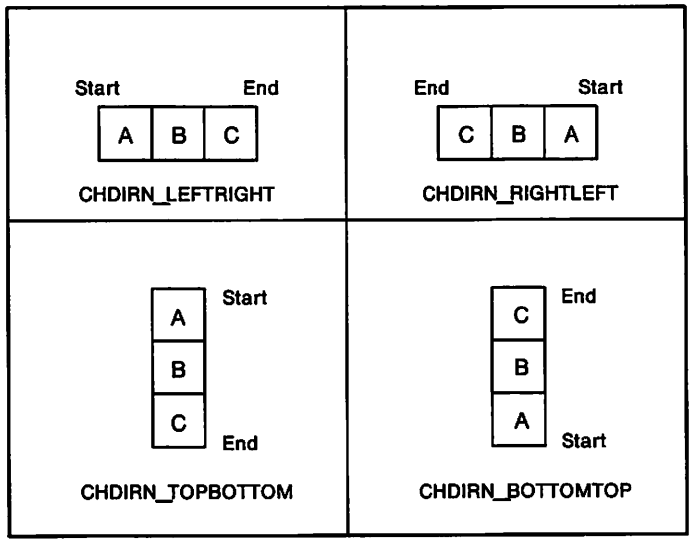


Figure 12-2. Character drawing directions, showing character origins.

Image Attributes

The device image attributes are bundled in a DIMAGEBUNDLE structure:

cAttr	Size of the attributes structure.
cDefs	This is set to zero as there is no IMAGEDEFS structure.
ibnd	IMAGEBUNDLE structure, see below.

IMAGEBUNDLE Mask: This mask is used in calls to GreDeviceSetAttributes to identify fields in the IMAGEBUNDLE structure. Valid flags and the fields that they identify are:

Flag	Field
IBB_COLOR	lColor
IBB_BACK_COLOR	lBackColor
IBB_MIX_MODE	usMixMode
IBB_BACK_MIX_MODE	usBackMixMode.

IMAGEBUNDLE Structure: The fields of an IMAGEBUNDLE structure are:

lColor Image foreground color

lBackColor Image background color

usMixMode Image foreground-mix mode

usBackMixMode Image background-mix mode.

Marker Attributes

The device marker attributes are bundled in a DMARKERBUNDLE structure:

cAttr	Size of MARKERBUNDLE structure.	
cDefs	Size of MARKERDEFS structure.	
mbnd	MARKERBUNDLE structure, see below.	
mdef	MARKERDEFS structure:	
	defSet	Marker set definition. If this value is passed as zero, the driver must use the default marker set. If the CDEF_GENERIC flag (below) is set, this is a long pointer to an engine font, otherwise it is a device-font identifier defined by the driver.
	fFlags	Valid flags are: **CDEF_GENERIC** Engine font (not device font) **CDEF_BOLD** Marker must be emboldened **CDEF_ITALIC** Marker must be italicized **CDEF_UNDERLINE** Marker must be underlined **CDEF_STRIKEOUT** Marker must be StrikeOut.
	CodePage	Code-page number.

MARKERBUNDLE Mask: This mask is used in calls to GreDeviceSetAttributes to identify fields in the MARKERBUNDLE structure. Valid flags and the fields that they identify are:

Flag	Field
MBB_COLOR	lColor
MBB_BACK_COLOR	lBackColor
MBB_MIX_MODE	usMixMode
MBB_BACK_MIX_MODE	usBackMixMode
MBB_SET	usSet
MBB_SYMBOL	usSymbol
MBB_BOX	sizfxCell.

MARKERBUNDLE Structure: The fields of a MARKERBUNDLE structure are:

lColor Marker foreground color.

lBackColor Marker background color.

usMixMode Marker foreground-mix mode.

usBackMixMode Marker background-mix mode.

usSet Specifies a local identifier (lcid) for the logical font:

Zero	Base marker set
Nonzero	Local identifier for the font identified in the mdef.defSet field of the marker-attributes bundle.

usSymbol Specifies the identity of the required marker symbol in the current marker set. If the value of usSymbol does not identify a marker in the current font, the standard default for the font is used. The default for the default font is a cross, for a loaded font it is the character identified by the FOCAMETRICS.usDefaultChar field.

All values in the range 0 through 255 are valid. The defined values for the default marker set are:

MARKSYM_CROSS	Cross
MARKSYM_PLUS	Plus
MARKSYM_DIAMOND	Diamond
MARKSYM_SQUARE	Square
MARKSYM_SIXPOINTSTAR	Six-point star
MARKSYM_EIGHTPOINTSTAR	Eight-point star
MARKSYM_SOLIDDIAMOND	Filled diamond
MARKSYM_SOLIDSQUARE	Filled square
MARKSYM_DOT	Dot
MARKSYM_SMALLCIRCLE	Small circle
MARKSYM_BLANK	Blank.

Note: MARKSYM_DEFAULT is the default value at the API. It is a reserved value and is not passed to the presentation driver.

sizfxCell Specifies fixed-point numbers for the width and height of a marker cell in world-coordinate space. This defines the background area for a marker.

Each dimension is represented as a signed 4-byte integer, with a notional binary point between bits 16 and 15. Thus: +2.5 is represented by 00028000H and −2.5 is represented by FFFD8000H.

The value of this attribute only affects the size of markers drawn with an outline (vector) font or marker set. Markers drawn from image (raster) sets are not affected.

```
(BOOL) GreDeviceGetAttributes (hdc, lPrimType, flAttrsMask, pAttrs, pInstance, lFunction)
```

Queries the attribute values currently set in the device.

Support

This function must be supported by the presentation driver.

The sample presentation driver handles this call by bit-map simulation. Before the call is passed to the display driver's dispatch table, the driver sets up the attribute bundle and caches the current settings. See module B0ATTRIB.C of the sample presentation driver.

Stack Frame

(HDC)	hdc	(device-context handle)
(LONG)	lPrimType	(bundle primitive type, see below)
(ULONG)	flAttrsMask	(attribute mask, see below)
(PBUNDLE)	pAttrs	(pointer to the fixed-format bundle record to which the attributes are returned. Fields other than those indicated by flAttrsMask must not be modified.)
(ULONG)	pInstance	(pointer to instance data)
(ULONG)	lFunction	(High-order word = Flags, Low-order word = NGreDeviceGetAttributes)

lPrimType Indicates the bundle type. Valid primitive values are:

PRIM_LINE Pen (line) attribute bundle, see page 12-52.
PRIM_CHAR Character attribute bundle, see page 12-56.
PRIM_MARKER Marker attribute bundle, see page 12-61.
PRIM_AREA Pattern attribute bundle, see page 12-54.
PRIM_IMAGE Image attribute bundle, see page 12-60.

flAttrsMask Specifies the attributes to be returned. This mask contains a bit corresponding to each attribute in the bundle record that is required. For each set bit, the handling routine must return the corresponding attribute values and default mask bits. Only the foreground-color and background-color attributes can be requested for any primitive type.

Return Codes

On completion, the handling routine must return BOOLEAN (fSuccess):

TRUE Successful.
FALSE Error.

When an error is detected, the handling routine must call WinSetErrorInfo to post the condition. Error codes for conditions that the handling routine is expected to check include:

PMERR_DEV_FUNC_NOT_INSTALLED
PMERR_INV_HDC.

GreGetPairKerningTable

```
(LONG) GreGetPairKerningTable (hdc, cKernPairs, pKernPairs, pInstance, lFunction)
```

Stores the kerning pairs of the current font to the buffer addressed by pKernPairs. The handling routine must transform all kerning-pair coordinates from device to world coordinates before handing the information to the calling routine, this can be done using GreConvert. When it is unable to do this, because the transform matrix is singular, it must log PMERR_COORDINATE_OVERFLOW.

Support

This function must be supported by the presentation driver.

The sample driver handles this call by bit-map simulation. The call parameters are passed unchanged to the display driver's dispatch table.

Stack Frame

(HDC)	hdc	(device-context handle)
(LONG)	cKernPairs	(number of kerning pairs, requested by the application)
(PKERNPAIRS)	pKernPairs	(pointer to kern pair records, see below)
(ULONG)	pInstance	(pointer to instance data)
(ULONG)	lFunction	(High-order word = Flags, Low-order word = NGreGetPairKerningTable)

pKernPairs KERNINGPAIRS structure:

usFirstChar	Code point for the first character.
usSecondChar	Code point for the second character.
sKerningAmount	2-byte signed integer, indicating the amount of kerning, positive numbers specify increased inter-character spacing.

Return Codes

On completion, the handling routine must return the number of kerning pairs returned in pKernPairs (cPairs) or GPI_ALTERROR when an error occurs.

When an error is detected, the handling routine must call WinSetErrorInfo to post the condition. Error codes for conditions that the handling routine is expected to check include:

PMERR_HDC_BUSY
PMERR_DEV_FUNC_NOT_INSTALLED
PMERR_COORDINATE_OVERFLOW
PMERR_INV_SETID
PMERR_INSUFFICIENT_MEMORY
PMERR_INV_LENGTH_OR_COUNT
PMERR_INV_CODEPAGE
PMERR_EXCEEDS_MAX_SEG_LENGTH
PMERR_INV_HDC
PMERR_INV_COORD_SPACE.

Remarks.

The number of kerning pairs is a field in the text metrics structure.

```
(BOOL) GreDeviceSetAttributes (hdc, lBType, flDefsMask, flAttrsMask, pAttrs, pInstance, lFunction)
```

Sets attributes in the attribute bundle specified by the lBType parameter. Pointers to the current attribute bundles are maintained by presentation drivers in the instance data structure; the handling routine for GreDeviceSetAttributes modifies the specified bundle as directed by flAttrsMask and flDefsMask parameters. See "Remarks." on page 12-66.

The handling routine must allow any attribute to be set to any value in the defined range for that attribute even when the value cannot be implemented on the device; for example, the driver for a vector printer or plotter device must accept BM_XOR background mix. When the driver is called to write to the device, it should map values that cannot be implemented to the default value.

If this call would set any of the attributes to a value that is not in the defined range of values for that attribute, the handling routine must restore all attributes to the value they had on entry to this routine.

When this function is called for the first time to set the character attributes, the handling routine should set the default font in the usSet parameter of the character attribute bundle, see page 12-56. If the default font is an engine font, the presentation driver must save the address and flags of the font. This ensures that the default font is restored if the DC is reset, see "Enable subfunction 09H — ResetDCState" on page 11-17.

Support

This function must be supported by the presentation driver.

The sample presentation driver handles this call by bit-map simulation. Before the call is passed to the display driver's dispatch table, the printer sets up the attribute bundle and caches the current settings. See module B0ATTRIB.C of the sample presentation driver

Stack Frame

(HDC)	hdc	(device-context handle)
(LONG)	lBType	(bundle type, see below)
(ULONG)	flDefsMask	(mask indicating the attributes to be set to their standard default values)
(ULONG)	flAttrsMask	(mask indicating the attributes to be modified)
(PBUNDLE)	pAttrs	(pointer to a bundle structure containing the new attribute values)
(ULONG)	pInstance	(pointer to instance data)
(ULONG)	lFunction	(High-order word = Flags, Low-order word = NGreDeviceSetAttributes)

lBType

Device attribute bundle The following device bundles are defined:

PRIM_AREA Pattern attribute bundle, see page 12-54.
PRIM_CHAR Character attribute bundle, see page 12-56.
PRIM_IMAGE Image attribute bundle, see page 12-60.
PRIM_LINE Line attribute bundle, see page 12-52.
PRIM_MARKER Marker attribute bundle, see page 12-61.

All device bundles share a similar format. They consist of two bundles, a bundle of logical attributes and a bundle of device information.

pAttrs Pointer to the DLINEBUNDLE, DCHARBUNDLE, DMARKERBUNDLE, DAREABUNDLE, or DIMAGEBUNDLE structure containing the new attributes.

GreDeviceSetAttributes

Return Codes

On completion, the handling routine must return BOOLEAN (fSuccess):

TRUE Successful.
FALSE Error.

When an error is detected, the handling routine must call WinSetErrorInfo to post the condition. Error codes for conditions that the handling routine is expected to check include:

PMERR_HDC_BUSY
PMERR_DEV_FUNC_NOT_INSTALLED
PMERR_COORDINATE_OVERFLOW
PMERR_INV_PATTERN_SET_ATTR
PMERR_INV_MIX_ATTR
PMERR_INV_BACKGROUND_MIX_ATTR
PMERR_INV_PATTERN_REF_PT_ATTR
PMERR_INV_LINE_TYPE_ATTR
PMERR_INV_CHAR_MODE_ATTR
PMERR_INV_CHAR_DIRECTION_ATTR
PMERR_INSUFFICIENT_MEMORY
PMERR_INV_LENGTH_OR_COUNT
PMERR_INV_CODEPAGE
PMERR_EXCEEDS_MAX_SEG_LENGTH
PMERR_INV_HDC
PMERR_INV_COORD_SPACE
PMERR_INV_PATTERN_SET_FONT
PMERR_HUGE_FONTS_NOT_SUPPORTED
PMERR_INV_COLOR_ATTR
PMERR_INV_BACKGROUND_COL_ATTR.

Remarks.

The parameters flAttrsMask and flDefsMask are instances of the mask for the specified attribute bundle; valid flags are listed under the bundle definitions on pages 12-52, 12-54, 12-56, 12-60, and 12-61. Flags set in flAttrsMask identify which fields in the attribute bundle are to be changed. For each flag that is set in flAttrsMask, the state of that flag in flDefsMask determines the source for the new value of the field: if the flag is set in both masks, the corresponding field should be set to its default value; if the flag is set in flAttrsMask and not set in flDefsMask, the corresponding field would be set from the relevant field in the bundle addressed by pAttrs.

In the attribute bundle addressed by pAttrs, the only fields that contain valid values are those that will be used to modify the device context's attribute bundle.

When setting pattern and area attributes, you must convert the pattern origin from world coordinates, in the attributes bundle, to device coordinates in the DC instance data.

```
(BOOL) GreDeviceSetGlobalAttribute (hdc, lAttrType, lAttribute, flOptions, pInstance, lFunction)
```

Sets the individual primitive attributes to the specified value in the line, area, character, image, and marker bundles.

If this call would set any of the attributes to a value that is not in the defined range of values for that attribute, the handling routine must restore all attributes to the value they had on entry to this routine.

Support

This function must be supported by the presentation driver.

The sample presentation driver handles this call by bit-map simulation. Before the call is passed to the display driver's dispatch table, the printer sets up the attribute bundle and caches the current settings. See module B0ATTRIB.C of the sample presentation driver.

Stack Frame

(HDC)	hdc	(device-context handle)
(LONG)	lAttrType	(specifies the attribute, see below)
(LONG)	lAttribute	(new attribute value)
(ULONG)	flOptions	(see below)
(ULONG)	pInstance	(pointer to instance data)
(ULONG)	lFunction	(High-order word = Flags, Low-order word = NGreDeviceSetGlobalAttribute)

lAttrType Attribute type:

ATYPE_COLOR Foreground color.
ATYPE_BACK_COLOR Background color.
ATYPE_MIX_MODE Foreground mix.
ATYPE_BACK_MIX_MODE Background mix.

Note: ATYPE_BACK_COLOR and ATYPE_BACK_MIX_MODE do not apply to the line bundle.

flOptions The only allowable option is GATTR_DEFAULT. When this flag is set, the attribute indicated by lAttrType is set to its default value. When this flag is not set, the attribute is set to the value of lAttribute.

Return Codes

On completion, the handling routine must return BOOLEAN (fSuccess):

TRUE Successful.
FALSE Error.

When an error is detected, the handling routine must call WinSetErrorInfo to post the condition. Error codes for conditions that the handling routine is expected to check include:

PMERR_DEV_FUNC_NOT_INSTALLED
PMERR_INV_MIX_ATTR
PMERR_INV_BACKGROUND_MIX_ATTR
PMERR_INV_HDC
PMERR_INV_COLOR_ATTR
PMERR_INV_BACKGROUND_COL_ATTR.

Device Functions 2

- GreNotifyClipChange
- GreNotifyTransformChange
- GreRealizeFont
- GreErasePS
- GreGetStyleRatio
- GreSetStyleRatio
- GreDeviceQueryFontAttributes
- GreDeviceQueryFonts.

```
(BOOL) GreNotifyClipChange (hdc, prcBound, cRect, idClipPath, pInstance, lFunction)
```

This function is called whenever there is any change to the DC region. The call gives the presentation driver an opportunity to optimize clipping by enumerating the clip rectangles and caching them whenever they change. Typically the handling routine would allocate more memory for the new DC region, when necessary, and call GreGetClipRects (page 14-62) to get the set of rectangles that define the new DC region. Also, for display devices only, the handling routine must clear the HDC_IS_DIRTY flag.

On completion the handling routine must redispatch this call to the graphics engine using the pointer supplied in the default dispatch table.

Support

This function must be supported by the presentation driver.

The sample presentation driver handles this call by bit-map simulation. The call parameters are passed unchanged to the display driver's dispatch table.

Stack Frame

(HDC)	hdc	(device-context handle)
(PRECTL)	prcBound	(pointer to a rectangle that bounds the new region, in screen coordinates)
(ULONG)	cRect	(number of rectangles in the new clip region, as returned by GetClipRects)
(ULONG)	idClipPath	(current ClipPath ID, see below)
(ULONG)	pInstance	(pointer to instance data)
(ULONG)	lFunction	(High-order word = Flags, Low-order word = NGreNotifyClipChange)

idClipPath For display devices, when idClipPath is passed as NCC_CLEANDC, the handling routine should clear the HDC_IS_DIRTY flag and return successful. See "VisRegionCallBack" on page 16-14 and "GreDeviceInvalidateVisRegion" on page 13-14.

Presentation drivers for other devices should ignore this parameter.

Return Codes

On completion, the handling routine must return BOOLEAN (fSuccess):

TRUE Successful.
FALSE Error.

Note: Drivers that do not cache the clip rectangles should, if there are no errors, return TRUE.

When an error is detected, the handling routine must call WinSetErrorInfo to post the condition. Error codes for conditions that the handling routine is expected to check include:

PMERR_HDC_BUSY
PMERR_DEV_FUNC_NOT_INSTALLED
PMERR_INV_RECT
PMERR_INV_COORDINATE
PMERR_INV_HDC
PMERR_INV_REGION_CONTROL.

GreNotifyTransformChange

```
(BOOL) GreNotifyTransformChange (hdc, flFlags, pXformdata, pInstance, lFunction)
```

Notifies the driver of a change in the transform from world to device coordinates. This function is called by the graphics engine to provide sufficient information to allow the device to optimize its calling of the GreConvert function, or, where possible, to make all point transformations itself.

The minimum requirement is for the handling routine to update the current position and pattern origin, and then call back to the GreNotifyTransformChange routine in the graphics engine. But note that:

- The handling routine must check that the new current position is valid.
- The handling routine must check that the change would not cause an overflow of the 16-bit coordinates for the device space.
- The handling routine must fail safe. If an error is detected or the call back to the engine fails, the routine must restore the initial values before returning FALSE.

Support

This function must be supported by the presentation driver.

The sample presentation driver handles this call by bit-map simulation. The call parameters are passed unchanged to the display driver's dispatch table.

Stack Frame

(HDC)	hdc	(device-context handle)
(ULONG)	flFlags	(see below)
(PNOTIFYTRANSFORMDATA)	pXformdata	(pointer to transform data structure, see below)
(ULONG)	pInstance	(pointer to instance data)
(ULONG)	lFunction	(High-order word = Flags, Low-order word = NGreNotifyTransformChange)

flFlags A set of flags that pass information about the complexity of the 2 by 2 matrix of the M11, M12, M21, and M22 components of the composite transform from world to device coordinates and show if a translation is required.

If the MATRIX_SIMPLE flag is not set, none of the other flags are valid. The MATRIX_X_NEGATE and MATRIX_Y_NEGATE flags are only meaningful when both MATRIX_SIMPLE and MATRIX_UNITS are set.

The flags are:

MATRIX_SIMPLE	Two entries are zero.
MATRIX_UNITS	All entries are +1 or −1.
MATRIX_XY_EXCHANGE	Zeros are on the diagonal.
MATRIX_X_NEGATE	X is hit by negative, this is only meaningful when MATRIX_UNITS is set.
MATRIX_Y_NEGATE	Y is hit by negative, this is only meaningful when MATRIX_UNITS is set.
MATRIX_TRANSLATION	Nonzero translation.

Examples:

```
Matrix = { 1.0, 0.0, 0.0, 1.0, 0, 0 } =>
     Flags = MATRIX_SIMPLE | MATRIX_UNITS
Matrix = { 1.0, 0.0, 0.0, 1.0, 5, 10 } =>
     Flags = MATRIX_SIMPLE | MATRIX_UNITS | MATRIX_TRANSLATION
Matrix = { 0.0, -1.0, 1.0, 0.0, 17, 5 } =>
     Flags = MATRIX_SIMPLE | MATRIX_UNITS | MATRIX_XY_EXCHANGE |
             MATRIX_Y_NEGATE | MATRIX_TRANSLATION
```

pXformdata Pointer to NOTIFYTRANSFORMDATA structure:

usType	Indicates fixed-point notation.
fxM11	Fixed-point matrix elements.
fxM12	..
fxM21	..
fxM22	..
lM41	Long translations.
lM42	..

Return Codes

On completion, the handling routine must return BOOLEAN (fSuccess):

TRUE	Successful.
FALSE	Error.

When an error is detected, the handling routine must call WinSetErrorInfo to post the condition. Error codes for conditions that the handling routine is expected to check include:

PMERR_BASE_ERROR
PMERR_INV_IN_AREA
PMERR_PATH_LIMIT_EXCEEDED
PMERR_INV_IN_PATH
PMERR_HDC_BUSY
PMERR_DEV_FUNC_NOT_INSTALLED
PMERR_COORDINATE_OVERFLOW
PMERR_INV_PATTERN_REF_PT_ATTR
PMERR_INV_LENGTH_OR_COUNT
PMERR_INV_HDC
PMERR_INV_COORD_SPACE
PMERR_INV_PICK_APERTURE_POSN.

GreRealizeFont

```
(ULONG) GreRealizeFont (hdc, cmdCommand, pLogFont, pFont, pInstance, lFunction)
```

Requests the driver to realize or delete a font.

Support

This function must be supported by the presentation driver.

The sample presentation driver handles this call by bit-map simulation. The call parameters are passed unchanged to the display driver's dispatch table.

Stack Frame

(HDC)	hdc	(device-context handle)
(ULONG)	cmdCommand	(see below)
(PFATTRS)	pLogFont	(logical font, see below)
(ULONG FAR *)	pFont	(see below)
(ULONG)	pInstance	(pointer to instance data)
(ULONG)	lFunction	(High-order word = Flags, Low-order word = NGreRealizeFont)

cmdCommand Valid commands are:

RF_DEVICE_FONT (1) The graphics engine asks the presentation driver if it is able to realize a device font for the logical font identified by pLogFont.

The action taken by the handling routine depends on the match number of the logical font. The match number is the lMatch parameter in the font metrics structure of the font specified. Font matching is discussed on page 15-45.

If lMatch is negative, the request is for the device font with the corresponding match number. The handling routine must return the font handle, or zero if no match was found.

If lMatch is zero, the handling routine must search the device fonts for an exact match to pLogFont, and return its handle, or zero if no match was found.

If lMatch is a positive value, the handling routine must return zero as if no font was found.

RF_LOAD_ENGINE_FONT (2) The presentation driver is requested to convert an engine font into a device font. The presentation driver must return the font handle, or zero when it is unable to convert the font.

RF_DELETE_FONT (3) The presentation driver is requested to delete a device font. The 32-bit font handle is passed in pFont, see 12-73.

RF_DELETE_ENGINE_FONT (4) This informs the presentation driver that a previously loaded engine font is being deleted. If font caching is not supported, the handling routine need take no action.

The font is passed to the presentation driver in the device character bundle during the GreDeviceSetAttribute call. (See "Character Attributes" on page 12-56.) A bit in the CHARDEFS structure indicates whether this is a long pointer to an engine font, or the handle to a device font.

pLogfont The initial value of cmdCommand determines the nature of this parameter:

RF_DEVICE_FONT	This is a long pointer to a logical font.
RF_LOAD_ENGINE_FONT	This is a long pointer to a logical font.
RF_DELETE_FONT	This is a long null pointer.
RF_DELETE_ENGINE_FONT	This is a long null pointer.

pFont The initial value of cmdCommand determines the nature of this parameter:

RF_DEVICE_FONT	This is a long null pointer.
RF_LOAD_ENGINE_FONT	This is a long pointer to an engine font.
RF_DELETE_FONT	This is a 32-bit device font handle.
RF_DELETE_ENGINE_FONT	This is a long pointer to an engine font.

Return Codes

The value returned by this function depends on whether the presentation driver is asked to realize or to delete the font:

When realizing, or loading a font, the handling routine must return a 32-bit logical-font handle, GPI_ALTERROR, or zero if the match, or load, was unsuccessful.

When deleting a font, the handling routine must return:

GPI_OK Successful.
GPI_ERROR Error.

When an error is detected, the handling routine must call WinSetErrorInfo to post the condition. Error codes for conditions that the handling routine is expected to check include:

PMERR_HDC_BUSY
PMERR_DEV_FUNC_NOT_INSTALLED
PMERR_COORDINATE_OVERFLOW
PMERR_INV_LENGTH_OR_COUNT
PMERR_INV_HDC
PMERR_INV_COORD_SPACE.

Remarks

GreRealizeFont is called by the graphics engine to allow the driver to satisfy logical font requests. The example on the following page shows a typical font-matching algorithm for the presentation driver.

```
/* If the facename is empty the application is asking for the  */
/* default font. Return NO_MATCH.                              */

if(Font->szFacename == NULL)
{
    return(0);
}

/* if the match number is positive then an engine font is     */
/* required. Return NO_MATCH.                                  */

if(Font->lfMatch > 0)
{
    return(0);
}

/* If the match number is negative, a device font is required. */
/* The driver should return the font if it exists or otherwise */
/* return NO_MATCH.                                            */

if(Font->lMatch < 0)
{
    if(font with required lMatch exists)
    {
        /* check the fsSelection flags and szFacename and if   */
        /* font is unable to satisfy these return NO_MATCH     */
        /* otherwise return the device font handle             */

        if((FATTR_FONTUSE_OUTLINE && font not outline)
        ||(FATTR_FONTUSE_TRANSFORMABLE && font not outline)
        ||(szFacename does not match font))
           return(0);
        else
          return(device_font_handle);
    }
    else
    {
        /* attempt metrics match (i.e. all metrics including   */
        /* szFacename, usCodePage, lAveCharWidth and           */
        /* lMaxBaselineExt) and if match exists return device  */
        /* font handle, otherwise return NO_MATCH              */

        if(metric match exists)
        {
            return(device_font_handle);
        }
        else
            return(0);
    }
}

/* The match number is zero. The driver should search for a    */
/* with the specified metrics and, if an exact match exists,   */
/* return the device font handle or otherwise return NO_MATCH  */

if(Font->lMatch == 0)
{
    if(metrics match exists - see above)
    {
        /* check the fsSelection flags and if                  */
        /* font is unable to satisfy these return NO_MATCH     */
        /* otherwise return the device font handle             */

        if((FATTR_FONTUSE_OUTLINE && font not outline)
        ||(FATTR_FONTUSE_TRANSFORMABLE && font not outline)
           return(0);
        else
          return(device_font_handle);
    }
    else
         return(0);
}
```

```
(BOOL) GreErasePS (hdc, pInstance, lFunction)
```

Resets the device context's presentation space to the background color (CLR_BACKGROUND). Presentation drivers for printers and plotters should return TRUE without taking any action.

The handling routine does not update GPI_BOUNDS or return correlation data. And it is not affected by the PCTL_DRAW control or COM_DRAW command flag. However, in presentation drivers for display devices, the handling routine should update USER_BOUNDS if the COM_ALT_BOUND command flag is set.

Support

This function must be supported by the presentation driver.

The sample presentation driver handles this call by bit-map simulation. The call parameters are passed unchanged to the display driver's dispatch table.

Stack Frame

(HDC)	hdc	(device-context handle)
(ULONG)	pInstance	(pointer to instance data)
(ULONG)	lFunction	(High-order word = Flags, Low-order word = NGreErasePS)

Return Codes

On completion, the handling routine must return BOOLEAN (fSuccess):

TRUE Successful.
FALSE Error.

When an error is detected, the handling routine must call WinSetErrorInfo to post the condition. Error codes for conditions that the handling routine is expected to check include:

PMERR_BASE_ERROR
PMERR_INV_IN_AREA
PMERR_INV_IN_PATH
PMERR_HDC_BUSY
PMERR_DEV_FUNC_NOT_INSTALLED
PMERR_COORDINATE_OVERFLOW
PMERR_BITMAP_NOT_SELECTED
PMERR_INV_PRIMITIVE_TYPE
PMERR_INV_SETID
PMERR_INV_CHAR_SET_ATTR
PMERR_INV_MARKER_SET_ATTR
PMERR_INV_PATTERN_SET_ATTR
PMERR_INV_MARKER_SYMBOL_ATTR
PMERR_UNSUPPORTED_ATTR
PMERR_INV_MIX_ATTR
PMERR_INV_BACKGROUND_MIX_ATTR
PMERR_INV_PATTERN_REF_PT_ATTR
PMERR_INV_LINE_WIDTH_ATTR
PMERR_INV_GEOM_LINE_WIDTH_ATTR
PMERR_INV_LINE_TYPE_ATTR
PMERR_INV_LINE_JOIN_ATTR
PMERR_INV_LINE_END_ATTR
PMERR_INV_CHAR_MODE_ATTR
PMERR_INV_CHAR_SHEAR_ATTR
PMERR_INV_CHAR_DIRECTION_ATTR
PMERR_UNSUPPORTED_ATTR_VALUE
PMERR_INSUFFICIENT_MEMORY
PMERR_INV_LENGTH_OR_COUNT
PMERR_INV_CODEPAGE

PMERR_EXCEEDS_MAX_SEG_LENGTH
PMERR_INV_RECT
PMERR_INV_COORDINATE
PMERR_INV_HDC
PMERR_INV_HRGN
PMERR_INV_DC_DATA
PMERR_INCOMPATIBLE_BITMAP
PMERR_INV_REGION_CONTROL
PMERR_INV_DRIVER_NAME
PMERR_INV_BITBLT_STYLE
PMERR_INV_BITBLT_MIX
PMERR_INV_HBITMAP
PMERR_INV_DC_TYPE
PMERR_INV_COORD_SPACE
PMERR_INV_ID
PMERR_BITMAP_IS_SELECTED
PMERR_HBITMAP_BUSY
PMERR_INV_USAGE_PARM
PMERR_INV_INFO_TABLE
PMERR_INV_BITMAP_DIMENSION
PMERR_INCORRECT_DC_TYPE
PMERR_REALIZE_NOT_SUPPORTED
PMERR_INV_COLOR_FORMAT
PMERR_INV_COLOR_DATA
PMERR_INV_COLOR_START_INDEX
PMERR_INV_COLOR_OPTIONS
PMERR_INV_COLOR_INDEX
PMERR_INV_SCAN_START
PMERR_INV_PICK_APERTURE_POSN
PMERR_INV_PATTERN_SET_FONT
PMERR_HUGE_FONTS_NOT_SUPPORTED
PMERR_INV_COLOR_ATTR
PMERR_INV_BACKGROUND_COL_ATTR.

Remarks

This function is subject to all clipping.

```
(BOOL) GreGetStyleRatio (hdc, pRatio, pInstance, lFunction)
```

Stores, at the location addressed by pRatio, the style ratio x and y direction step values.

When the line type is LINETYPE_ALTERNATE, the handling routine must restore a value of zero for the style ratio, to ensure that the style mask is rotated after each pel is drawn. See "Line and Curve Styling" on page 12-2.

Support

This function must be supported by the presentation driver.

See module B0ATTRIB.C of the sample presentation driver.

Stack Frame

(HDC)	hdc	(device-context handle)
(PBYTE)	pRatio	(pointer to style-ratio value, see below)
(ULONG)	pInstance	(pointer to instance data)
(ULONG)	lFunction	(High-order word = Flags, Low-order word = NGreGetStyleRatio)

pRatio The style ratio is defined as a two-byte value, the low-order byte indicates a step in the x-direction, and the high-order byte a step in the y-direction.

Return Codes

This function returns BOOLEAN (fSuccess):

TRUE Successful.
FALSE Error.

When an error is detected, the handling routine must call WinSetErrorInfo to post the condition. Error codes for conditions that the handling routine is expected to check include:

PMERR_DEV_FUNC_NOT_INSTALLED
PMERR_INV_HDC.

GreSetStyleRatio

```
(BOOL) GreSetStyleRatio (hdc, pRatio, pInstance, lFunction)
```

Sets the style ratio used by the driver's line-drawing algorithm to determine which pels should be set on for a sloping line. Display drivers must support this function so that printer drivers, whose device may have a different style ratio, can use the display driver to draw into a bit map that the printer driver can use.

Support

This function must be supported by the presentation driver.

The sample driver handles this call by bit-map simulation. Before passing the call to the display driver, the handling routine checks the origin of the call and the validity of pRatio. See module B0ATTRIB.C of the sample presentation driver.

Stack Frame

(HDC)	hdc	(device-context handle)
(PBYTE)	pRatio	(pointer to two unsigned bytes, corresponding to the aspect of the pels that a line is drawn on)
(ULONG)	pInstance	(pointer to instance data)
(ULONG)	lFunction	(High-order word = Flags, Low-order word = NGreSetStyleRatio)

pRatio The style ratio is defined as a two-byte value, the low-order byte indicates a step in the x-direction, and the high-order byte a step in the y-direction. Typical values for style ratios are:

- EGA devices:
 x-direction 64
 y-direction 85
- One-to-one devices:
 x-direction 64
 y-direction 64

 In this instance, the style ratio steps are set to 64:64 rather than 1:1 to ensure that a single dot in a line-style pattern is a sensible length. The length of a single dot in the pattern is (256/step_value) pels.

Return Codes

On completion, the handling routine must return BOOLEAN (fSuccess):

TRUE Successful.
FALSE Error.

When an error is detected, the handling routine must call WinSetErrorInfo to post the condition. Error codes for conditions that the handling routine is expected to check include:

PMERR_DEV_FUNC_NOT_INSTALLED
PMERR_INV_HDC.

```
(BOOL) GreDeviceQueryFontAttributes (hdc, cMetric, pfmMetrics, pInstance, lFunction)
```

Stores the metrics of the currently-selected font, at the location addressed by pfmMetrics.

The handling routine must transform device coordinates to world coordinates before returning the results to the calling routine. This can be done using GreConvert.

Support

This function must be supported by the presentation driver.

The sample driver handles this call by bit-map simulation. The call parameters are passed unchanged to the display driver's dispatch table.

Stack Frame

(HDC)	hdc	(device-context handle)
(ULONG)	cMetrics	(size of FONTMETRICS structure)
(PFONTMETRICS)	pfmMetrics	(pointer to FONTMETRICS structure)
(ULONG)	pInstance	(pointer to instance data)
(ULONG)	lFunction	(High-order word = Flags, Low-order word = NGreDeviceQueryFontAttributes)

Return Codes

On completion, the handling routine must return BOOLEAN (fSuccess):

TRUE Successful.
FALSE Error.

When an error is detected, the handling routine must call WinSetErrorInfo to post the condition. Error codes for conditions that the handling routine is expected to check include:

PMERR_DEV_FUNC_NOT_INSTALLED
PMERR_INV_HDC.

GreDeviceQueryFonts

```
(LONG) GreDeviceQueryFonts (hdc, flOptions, pszFaceName, pfmMetrics, cMetrics, pcFonts, pInstance, lFunction)
```

If the QF_PUBLIC option flag (see below) is set, GreDeviceQueryFonts returns the characteristics of device fonts in an array of FONTMETRICS structures. The returned fonts should include those that correspond to device modes such as 'expanded' and 'expanded-bold'. And, when the DC is not set to draft mode, the returned fonts should be those that can be positioned to the nearest pel (such precision is not necessary if the DC is in draft mode; draft mode is set by the system calling the DEVESC_DRAFTMODE subfunction of GreEscape).

In the FONTMETRICS structures, the handling routine should :

- Set the lMatch field to a negative value. (This allows the driver to map the font when the value is specified in call to the GreRealizeFont routine.)
- Set the szFacename field to a meaningful name (for example, 'Expanded' or 'Expanded bold ').
- Set the usCodepage field to zero. (This field has no significance in this context.)

The presentation driver must transform device coordinates to world coordinates before it returns the results to the calling routine. This can be done using GreConvert.

For drivers that support only outline fonts, the return values should be for outline fonts even when image fonts have been loaded.

Support

This function must be supported by the presentation driver.

The sample driver handles this call by bit-map simulation. The call parameters are passed unchanged to the display driver's dispatch table.

Stack Frame

(HDC)	hdc	(device-context handle)
(ULONG)	flOptions	(option flags, see below)
(PSZ)	pszFacename	(pointer to facename to match; if this is a null pointer, all faces are matched)
(PFONTMETRICS)	pfmMetrics	(pointer to array of FONTMETRICS structures)
(LONG)	cMetrics	(number of bytes of each metrics structure in the metrics array)
(PLONG)	pcFonts	(pointer to the number of fonts requested, see below)
(ULONG)	pInstance	(pointer to instance data)
(ULONG)	lFunction	(High-order word = Flags, Low-order word = NGreDeviceQueryFonts)

flOptions The only valid flag for this function is:

QF_PUBLIC When this flag is set, the handling routine must return all device fonts. (Device fonts are public fonts and they are returned in the array addressed by pfmMetrics.) If this flag is not set, the handling routine should not return any fonts.

pcFonts This is a pointer to the number of fonts requested. On completion, the handling routine modifies the value indicated to the number of fonts returned.

An application can determine the number of public fonts available to it by passing a value of zero at the address indicated by this pointer.

Return Codes

The handling routine should return the number of fonts not returned or GPI_ALTERROR if an error occurred.

When an error is detected, the handling routine must call WinSetErrorInfo to post the condition. Error codes for conditions that the handling routine is expected to check include:

PMERR_DEV_FUNC_NOT_INSTALLED
PMERR_INV_HDC.

Device Functions 3

- GreResetBounds
- GreGetBoundsData
- GreAccumulateBounds
- GreGetCodePage
- GreSetCodePage
- GreLockDevice
- GreUnlockDevice
- GreGetLineOrigin
- GreSetLineOrigin.

GreResetBounds

```
(BOOL) GreResetBounds (hdc, flOptions, pInstance, lFunction)
```

Resets the bounds to their initial values: 07FFFFFH for the minimum coordinates and F800000H for the maximum coordinates.

Presentation drivers for printer and plotter devices are only required to support GPI bounds. Drivers for displays must also support USER bounds for the window manager.

Support

This function must be supported by the presentation driver.

The sample presentation driver handles this call by bit-map simulation. The call parameters are passed unchanged to the display driver's dispatch table.

Stack Frame

(HDC)	hdc	(device-context handle)
(ULONG)	flOptions	(option flags, see below)
(ULONG)	pInstance.	(pointer to instance data)
(ULONG)	lFunction	(High-order word = Flags, Low-order word = NGreResetBounds)

flOptions Valid flags are:

RB_GPI Reset the GPI bounds.
RB_USER Reset the USER bounds.

Return Codes

This function returns BOOLEAN (fSuccess):

TRUE Successful.
FALSE Error.

When an error is detected, the handling routine must call WinSetErrorInfo to post the condition. Error codes for conditions that the handling routine is expected to check include:

PMERR_DEV_FUNC_NOT_INSTALLED
PMERR_INV_HDC.

```
(BOOL) GreGetBoundsData (hdc, flOptions, pBoundsData, pInstance, lFunction)
```

Stores the bounding rectangle of previous drawing primitives at the address indicated by pBoundsData.

All presentation drivers must support GPI bounds; these bounds should be transformed to model-space coordinates when they are accumulated. Presentation drivers for display devices must also support USER bounds in screen coordinates.

Bounds are inclusive. A null boundary is represented by the minimum coordinates of the rectangle being greater than its maximum coordinates.

If the bounds have been reset, a null value is returned for pBoundsData.

Support

This function must be supported by the presentation driver.

The sample presentation driver handles this call by bit-map simulation. The call parameters are passed unchanged to the display driver's dispatch table.

Stack Frame

(HDC)	hdc	(device-context handle)
(ULONG)	flOptions	(option flags, see below)
(PRECTL)	pBoundsData	(pointer to the address for the returned RECTL structure that defines the specified bounding rectangle)
(ULONG)	pInstance	(pointer to instance data)
(ULONG)	lFunction	(High-order word = Flags, Low-order word = NGreGetBoundsData)

flOptions The option flags define which bounding rectangle the handling routine should return. Valid values are:

GBD_GPI Return GPI bounds in model-space coordinates.
GBD_USER Return current USER bounds, in screen coordinates, and reset USER bounds to their initial value.

Return Codes

This function returns BOOLEAN (fSuccess):

TRUE Successful.
FALSE Error.

When an error is detected, the handling routine must call WinSetErrorInfo to post the condition. Error codes for conditions that the handling routine is expected to check include:

PMERR_DEV_FUNC_NOT_INSTALLED
PMERR_INV_HDC.

GreAccumulateBounds

```
(BOOL) GreAccumulateBounds (hdc, prcRect, pInstance, lFunction)
```

This function is called to merge bounds into the total bounds held by the presentation driver.

The presentation driver does bounds calculations for all drawing primitives. When accumulating, it must convert the bounds to model space, as they are accumulated, before merging with the GPI bounds. This can be done with GreConvert.

GreResetBounds (page 12-82) and GreGetBoundsData (page 12-83) are also related to this function.

Support

This function must be supported by the presentation driver.

The sample presentation driver handles this call by bit-map simulation. The call parameters are passed unchanged to the display driver's dispatch table.

Stack Frame

(HDC)	hdc	(device-context handle)
(PRECTL)	prcRect	(pointer to rectangle, defined as a RECTL structure in device coordinates)
(ULONG)	pInstance	(pointer to instance data)
(ULONG)	lFunction	(High-order word = Flags, Low-order word = NGreAccumulateBounds)

Return Codes

This function returns BOOLEAN (fSuccess):

TRUE Successful.
FALSE Error.

When an error is detected, the handling routine must call WinSetErrorInfo to post the condition. Error codes for conditions that the handling routine is expected to check include:

PMERR_HDC_BUSY
PMERR_DEV_FUNC_NOT_INSTALLED
PMERR_COORDINATE_OVERFLOW
PMERR_INV_LENGTH_OR_COUNT
PMERR_INV_RECT
PMERR_INV_HDC
PMERR_INV_COORD_SPACE.

```
(long) GreGetCodePage (hdc, pInstance, lFunction)
```

Returns the current code page. This is the default code page, obtained by WinQueryProcessCp during the enabling of the DC (page 11-12) or that set by GreSetCodePage. This function applies to the default font, not the currently selected font, which you can determine with a call to GreQueryFontAttributes (page 15-53).

Support

This function must be supported by the presentation driver.

The sample presentation driver handles this call by bit-map simulation. The call parameters are passed unchanged to the display driver's dispatch table.

Stack Frame

(HDC)	hdc	(device-context handle)
(ULONG)	pInstance	(pointer to instance data)
(ULONG)	lFunction	(High-order word = Flags, Low-order word = NGreGetCodePage)

Return Codes

On completion, the handling routine must return the current code page (lCodePage) or GPI_ERROR.

When an error is detected, the handling routine must call WinSetErrorInfo to post the condition. Error codes for conditions that the handling routine is expected to check include:

PMERR_DEV_FUNC_NOT_INSTALLED.

GreSetCodePage

```
(BOOL) GreSetCodePage (hdc, lCodePage, pInstance, lFunction)
```

Sets the current code page for characters written with the base (default) font. The default is the font that the system uses when the cbnd.usSet attribute is 0000H. See "Character Attributes" on page 12-56. When the base font is not in use, lCodePage is saved until required. When a DC is initialized, the codepage is set to the default codepage obtained from WinQueryProcessCp.

Support

This function must be supported by the presentation driver.

The sample presentation driver handles this call by bit-map simulation. The call parameters are passed unchanged to the display driver's dispatch table.

Stack Frame

(HDC)	hdc	(device-context handle)
(ULONG)	lCodePage	(new code page)
(ULONG)	pInstance	(pointer to instance data)
(ULONG)	lFunction	(High-order word = Flags, Low-order word = NGreSetCodePage)

Return Codes

On completion, the handling routine must return BOOLEAN (fSuccess):

TRUE Successful.
FALSE Error.

When an error is detected, the handling routine must call WinSetErrorInfo to post the condition. Error codes for conditions that the handling routine is expected to check include:

PMERR_INV_IN_AREA
PMERR_DEV_FUNC_NOT_INSTALLED
PMERR_INSUFFICIENT_MEMORY
PMERR_INV_CODEPAGE
PMERR_EXCEEDS_MAX_SEG_LENGTH
PMERR_INV_HDC.

```
(BOOL) GreLockDevice (hdc, pInstance, lFunction)
```

Locks a device for use by a single thread.

Support

This function must be supported all presentation drivers. For hard-copy devices, the driver need do nothing except return successful. See module A0CONTRL.C of the sample presentation driver.

Stack Frame

(HDC)	hdc	(device-context handle)
(ULONG)	pInstance	(pointer to instance data)
(ULONG)	lFunction	(High-order word = Flags, Low-order word = NGreLockDevice)

Return Codes

On completion, the handling routine must return BOOLEAN (fSuccess):

TRUE Successful.
FALSE Error.

When an error is detected, the handling routine must call WinSetErrorInfo to post the condition. Error codes for conditions that the handling routine is expected to check include:

PMERR_DEV_FUNC_NOT_INSTALLED
PMERR_INV_HDC.

Remarks

This function synchronizes the use and update of the visible region by allowing all current and pending drawing to finish, and then blocking any requests to draw from other threads until GreUnlockDevice is called. On exit, the only thread allowed to continue with screen operations is the one that acquires the lock.

To prevent deadlock, GreDeath cannot be called while the visible region is locked.

GreUnlockDevice

```
(BOOL) GreUnlockDevice (hdc, pInstance, lFunction)
```

Allows all pending screen input/output operations, blocked by GreLockDevice, to continue.

Support

This function must be supported all presentation drivers. For hard-copy devices, the driver need do nothing except return successful. See module A0CONTRL.C of the sample presentation driver.

Stack Frame

(HDC)	hdc	(device-context handle)
(ULONG)	pInstance	(pointer to instance data)
(ULONG)	lFunction	(High-order word = Flags, Low-order word = NGreUnlockDevice)

Return Codes

On completion, the handling routine must return BOOLEAN (fSuccess):

TRUE Successful.
FALSE Error.

When an error is detected, the handling routine must call WinSetErrorInfo to post the condition. Error codes for conditions that the handling routine is expected to check include:

PMERR_DEV_FUNC_NOT_INSTALLED.

Remarks

This function is used to synchronize the use and update of visible region.

```
(LONG) GreGetLineOrigin (hdc, pptXY, pInstance, lFunction)
```

Returns the current line style from the DC instance data and stores the current position to the address indicated by pptXY. The presentation driver maintains the line-style information.

Some lines and curves can be drawn either by the presentation driver or by simulations that must be able to query and set the style as required.

The high-order word of the style number contains the first/last pel information. If the value of this byte is zero the first pel is not drawn. See "First and Last Pel Considerations" on page 12-3. The low-order byte indicates the position in the style mask. This is a count, held in the three least significant bits, of the number of positions that the style mask byte is rotated. The next byte is the state of the style error value.

Support

This function must be supported by the presentation driver.

The sample presentation driver handles this call by bit-map simulation. The call parameters are passed unchanged to the display driver's dispatch table.

Stack Frame

(HDC)	hdc	(device-context handle)
(PPOINTL)	pptXY	(pointer to an (x,y) coordinate pair, to which the current position is returned)
(ULONG)	pInstance	(pointer to instance data)
(ULONG)	lFunction	(High-order word = Flags, Low-order word = NGreGetLineOrigin)

Return Codes

On completion, this function returns the style number (lStyle), or GPI_ALTERROR when an error occurs.

When an error is detected, the handling routine must call WinSetErrorInfo to post the condition. Error codes for conditions that the handling routine is expected to check include:

PMERR_DEV_FUNC_NOT_INSTALLED
PMERR_INV_HDC.

GreSetLineOrigin

```
(BOOL) GreSetLineOrigin (hdc, pptXY, lStyle, pInstance, lFunction)
```

Sets the current line style and current position, in world coordinates. The new line style is stored in the DC instance data structure.

Some lines and curves can be drawn either by the presentation driver or by simulations, and so, must be able to query and set the style as required.

The high-order word of the style number contains the first/last pel information. The low-order byte indicates the position in the style mask. The next byte is the state of the style error value. See "GreGetLineOrigin" on page 12-89.

Support

This function must be supported by the presentation driver.

The sample presentation driver handles this call by bit-map simulation. The call parameters are passed unchanged to the display driver's dispatch table.

Stack Frame

(HDC)	hdc	(device-context handle)
(PPOINTL)	pptXY	(pointer to an (x,y) coordinate pair, to which the current position is returned)
(ULONG)	lStyle	(style number)
(ULONG)	pInstance	(pointer to instance data)
(ULONG)	lFunction	(High-order word = Flags, Low-order word = NGreSetLineOrigin)

Return Codes

This function returns BOOLEAN (fSuccess):

TRUE Successful.
FALSE Error.

When an error is detected, the handling routine must call WinSetErrorInfo to post the condition. Error codes for conditions that the handling routine is expected to check include:

PMERR_BASE_ERROR
PMERR_PATH_LIMIT_EXCEEDED
PMERR_HDC_BUSY
PMERR_DEV_FUNC_NOT_INSTALLED
PMERR_COORDINATE_OVERFLOW
PMERR_INV_LENGTH_OR_COUNT
PMERR_INV_HDC
PMERR_INV_COORD_SPACE.

Example

This C language example outlines a strategy by which your handling routine could use the information from a GreSetLineOrigin call.

```
/*********************************************************************/
/**                                                                 **/
/**  StyleMask is a single byte.                                    **/
/**  usState is a USHORT with the error byte as the low-order       **/
/**  byte and the mask position as the high-order byte.  The        **/
/**  three low-order bits of the mask position represent the        **/
/**  number of bits by which StyleMask has been rotated.            **/
/**                                                                 **/
/*********************************************************************/

    while (necessary--)

         {

/*********************************************************************/
/**                                                                 **/
/**  Do we need to draw the first pel in the line?                  **/
/**                                                                 **/
/*********************************************************************/

         if (stylemask & 0x80)
             SetPel (x,y);

/*********************************************************************/
/**                                                                 **/
/**  Save the current style state.                                  **/
/**                                                                 **/
/*********************************************************************/

         usStateOld=usState;
         switch (LineMajor)
             {
             case yMajor:
                  usState=usState+yRatio;
                  break;
             default:
                  usState=usState+xRatio;
                  break;
             }

          if HIBYTE (usState) != HIBYTE (usStateOld)

/*********************************************************************/
/**                                                                 **/
/**  If the error byte has overflowed, rotate the style ratio.      **/
/**                                                                 **/
/**  The style mask is reset every eighth rotation.                 **/
/**                                                                 **/
/*********************************************************************/

            RotateLeftOne (StyleRatio);
          UpDateNext (x,y);
         }

/*********************************************************************/
/**                                                                 **/
/*********************************************************************/
```

The values for pRatio are set and queried by SetStyleRatio and GetStyleRatio respectively (see pages 12-78 and 12-77). Alternatively, you can specify your own device-specific defaults for xRatio and yRatio.

Color Functions

The functions described in this section are:

- GreQueryColorData
- GreQueryLogColorTable
- GreCreateLogColorTable
- GreRealizeColorTable
- GreUnrealizeColorTable
- GreQueryRealColors
- GreQueryNearestColor
- GreQueryColorIndex
- GreQueryRGBColor.

The Logical Color Table

By default, the color mode for a DC is set to index mode and the DC has a logical color table set to the values given below. When in index mode, these defaults are always considered to be part of the color table unless they are explicitly overwritten by CreateLogColorTable (see page 12-96).

Note: Presentation drivers that support less than 16 colors must map the values 0 (CLR_BACKGROUND) through 15 (CLR_PALEGRAY) to device colors. If the GreQueryColorData routine is called while the default color table is the current color table, the presentation driver should return the device colors.

Default values for the logical color table are:

CLR_FALSE	(−5) All color planes, or bits, or both, are FALSE.
CLR_TRUE	(−4) All color planes, or bits, or both, are TRUE.
CLR_DEFAULT	(−3) This is the API default. It is a reserved value and is not passed to the presentation driver.
CLR_WHITE	(−2) This index is never loaded explicitly. It always produces white when the default table is in force or when the index is set to RGB. When, with a realized color table and an index that is not RGB, this option is unavailable, it produces CLR_BACKGROUND.
CLR_BLACK	(−1) This index is never loaded explicitly. It always produces black when the default table is in force or when the index is set to RGB. When, with a realized color table and an index that is not RGB, this option is unavailable, it produces CLR_NEUTRAL.
CLR_BACKGROUND	(0) This is the natural background color of the device. For a printer, it is the paper color and for a display device it is the default window color (SYSCLR_WINDOW).
CLR_BLUE	(1)
CLR_RED	(2)
CLR_PINK	(3)
CLR_GREEN	(4)
CLR_CYAN	(5)
CLR_YELLOW	(6)
CLR_NEUTRAL	(7) This is a device-dependent contrasting color. For a display device it is the default window text color (SYSCLR_WINDOWTEXT).
CLR_DARKGRAY	(8)
CLR_DARKBLUE	(9)
CLR_DARKRED	(10)
CLR_DARKPINK	(11)
CLR_DARKGREEN	(12)
CLR_DARKCYAN	(13)
CLR_BROWN	(14)
CLR_PALEGRAY	(15)

Colors with indexes greater than 15 are device-dependent defaults that must be defined by the presentation driver. The effective range of the color table, which includes the default color table, is −5 through MaxIndex. Color indexes outside this range, which have not been loaded, should not be used by applications as these colors cannot be guaranteed.

The default colors are always available, where this is physically possible, on a device. For devices that support more than 16 colors, requested colors can be mapped to colors other than the defaults, when they exist. Such colors cannot be guaranteed to be similar for different devices, they can also be different for other releases of applications and presentation drivers. Applications that depend on precise colors beyond the defaults, must query the available colors (see "GreQueryRealColors" on page 12-100), and, when necessary, realize their own color tables (see "GreRealizeColorTable" on page 12-98).

Support for Monochrome Devices

Presentation drivers for monochrome devices must be able to draw pictures intended for color devices.

A typical solution for printers and plotters is to map:

- Background color to paper color.
- Foreground color to printer foreground, except when:
 1. In RGB mode: if the foreground RGB matches the default background RGB, use paper color.
 2. In index mode,: if the RGB foreground index matches the RGB color for index 0, use paper color.

To map the RGB colors to the device, the presentation driver must first establish the reset color. This is the base color for the device and can be:

- Paper color. For a printer with no loaded color table.
- SYSCLR_WINDOW. For a monochrome display with no loaded color table.
- CLR_BACKGROUND. For any monochrome device that has a loaded color table.

Any color that is not the reset color, is considered to be the contrast color. The values for the reset color and contrast color are either 000000H or FFFFFFH, so that, when the reset color is 000000H, the contrast color is FFFFFFH.

CLR_TRUE, CLR_FALSE, and CLR_DEFAULT are always honored independently of the reset color. The interpretation of CLR_BLACK and CLR_WHITE depends on the reset color.

When GreQueryNearestColor is called for a monochrome device, the value returned is either the reset color or the contrast color.

GreErasePS causes the color to be set to the value of the reset color. See "GreErasePS" on page 12-75.

GreBitblt can also be used to transfer a color bit map to a monochrome device, or bit map. When this happens, the source image background color becomes the reset color and all other pels are represented by the contrast color. See "GreBitblt" on page 12-35.

GreQueryColorData

```
(BOOL) GreQueryColorData (hdc, cArray, pArray, pInstance, lFunction)
```

Stores, at the location addressed by pArray, an array of information about the currently available logical color table and device colors.

When the current table is the default logical color table, presentation drivers that support less than 16 colors should return the device colors that the 16 colors from 0 (CLR_BACKGROUND) through 15 (CLR_PALEGRAY) have been mapped to.

Support

This function must be supported by the presentation driver.

The sample driver handles this call by bit-map simulation. The call parameters are passed unchanged to the display driver's dispatch table.

Stack Frame

(HDC)	hdc	(device-context handle)
(LONG)	cArray	(number of elements supplied in Array)
(PLONG)	pArray	(pointer to array, see below))
(ULONG)	pInstance	(pointer to instance data)
(ULONG)	lFunction	(High-order word = Flags, Low-order word = NGreQueryColorData)

pArray On return, this array contains:

lArray[QCD_LCT_FORMAT] Format of loaded color table (if any):
- **LCOLF_DEFAULT** Default color table is in force.
- **LCOLF_INDRGB** Color table loaded, which provides translation from index to RGB.
- **LCOLF_RGB** Color index = RGB.

lArray[QCD_LCT_LOINDEX] Smallest color index loaded (this is always 0).
lArray[QCD_LCT_HIINDEX] Largest color index loaded (this is never less than 15).

Information is only returned for the number of elements supplied, any extra elements supplied must be zeroed by the handling routine.

Return Codes

On completion, the handling routine must return BOOLEAN (fSuccess):

TRUE Successful.
FALSE Error.

When an error is detected, the handling routine must call WinSetErrorInfo to post the condition. Error codes for conditions that the handling routine is expected to check include:

PMERR_DEV_FUNC_NOT_INSTALLED
PMERR_INV_LENGTH_OR_COUNT
PMERR_INV_HDC.

```
(LONG) GreQueryLogColorTable (hdc, flOptions, iStart, cArray, pArray, pInstance, lFunction)
```

Stores, at the location addressed by pArray, an array of the current logical color values.

Support

This function must be supported by the presentation driver.

The sample driver handles this call by bit-map simulation. The call parameters are passed unchanged to the display driver's dispatch table.

Stack Frame

(HDC)	hdc	(device-context handle)
(ULONG)	flOptions	(see below)
(LONG)	iStart	(starting index for which data is to be returned)
(LONG)	cArray	(number of elements available in the array)
(PLONG)	pArray	(pointer to the array in which the information is returned)
(ULONG)	pInstance	(pointer to instance data)
(ULONG)	lFunction	(High-order word = Flags, Low-order word = NGreQueryLogColorTable)

flOptions The only valid option is:

LCOLOPT_INDEX If set, the handling routine must return the index for each RGB value.

Other flags are reserved and must be zero.

pArray When LCOLOPT_INDEX is not set, this points to an array of RGB color values in which the information is returned. Each value is as defined for CreateLogColorTable on page 12-96, starting with the specified index and ending either when there are no further loaded entries in the table, or when cCount has been exhausted. If the logical color table is not loaded with a contiguous set of indexes, CLR_NOINDEX is returned as the color value for any index values that are not loaded.

When LCOLOPT_INDEX is set, this is an array of alternating color indexes and values, in the order index1, value1, index2, value2... If the logical color table is not loaded with a contiguous set of indexes, any index values that are not loaded are skipped.

Return Codes

The handling routine must return a LONG value indicating the number of elements returned in pArray or:

QLCT_ERROR Error.
QLCT_RGB Color table is in RGB mode and no elements are returned.

When an error is detected, the handling routine must call WinSetErrorInfo to post the condition. Error codes for conditions that the handling routine is expected to check include:

PMERR_DEV_FUNC_NOT_INSTALLED
PMERR_INV_HDC
PMERR_INV_LENGTH_OR_COUNT
PMERR_INV_COLOR_OPTIONS
PMERR_INV_COLOR_START_INDEX.

GreCreateLogColorTable

```
(BOOL) GreCreateLogColorTable (hdc, flOptions, lFormat, iStart, cCount,
                               pData, pInstance, lFunction)
```

Defines the entries of the logical color table.

Support

This function must be supported by the presentation driver.

The sample driver handles this call by bit-map simulation. Before passing the call to the display driver, the handling routine must check that the value of flOptions is valid. When the display driver returns, the printer driver must remap the colors in the attribute bundles. See module K0COLORS.C of the sample driver.

Stack Frame

(HDC)	hdc	(device-context handle)
(ULONG)	flOptions	(see below)
(LONG)	lFormat	(format of entries in the table, see below)
(LONG)	iStart	(starting index, only relevant for LCOLF_CONSECRGB)
(LONG)	cCount	(number of elements supplied in application data area, see below)
(PLONG)	pData	(pointer to application data area, see below)
(ULONG)	pInstance	(pointer to instance data)
(ULONG)	lFunction	(High-order word = Flags, Low-order word = NGreCreateLogColorTable)

flOptions Valid options are:

LCOL_RESET This flag is set to indicate that the handling routine must reset the color table to default, before processing the remainder of this function.

Note: This option is assumed when the color table is changed from LCOLF_RGB to LCOLF_INDRGB or LCOLF_CONSECRGB.

LCOL_REALIZABLE This flag is set to indicate to the handling routine that the application can call GreRealizeColorTable at the appropriate time. This can affect the way the handling routine maps the indexes when the logical color table is not realized. A realizable color table is only required to provide color mapping for its color indexes while it is realized.

If this flag is not set, GreRealizeColorTable has no effect and posts a warning.

LCOL_PURECOLOR For solid patterns, pattern colors that are not available can be approximated by dithering. When this flag is set, only pure colors are used; the handling routine must not dither colors. The default is to allow color dithering.

Other flags are reserved and must be zero.

lFormat Valid formats are:

LCOLF_INDRGB Array of (index, RGB) values. Each pair of values contains 8 bytes, a 4-byte index and a 4-byte color. This sets the color table into index mode (and forces LCOL_RESET) if it is in RGB mode.

LCOLF_CONSECRGB Array of (RGB) values corresponding to color indexes, starting from lStart upwards. Each entry is a 4-byte value. This sets the color table into index mode (and forces LCOL_RESET) if it is in RGB mode.

LCOLF_RGB Color index = RGB. This sets the color table to RGB mode.

cCount the number of elements supplied in pData.

This may be set to zero if, for instance, the color table is merely to be reset to the default, or LCOLF_RGB. When this is zero, LCOLF_INDRGB and LCOLF_CONSECRGB have the same effect.

For LCOLF_INDRGB it must be an even number.

pData data area containing the color table definition data; the format depends on the value of lFormat.

Each color value is a 4-byte integer with a value of

```
(R * 65536) + (G * 256) + B
```

where:

R = red intensity value
G = green intensity value
B = blue intensity value.

The maximum intensity for each primary is 255.

Return Codes

On completion, the handling routine must return BOOLEAN (fSuccess):

TRUE Successful.
FALSE Error.

Error checking for this function is performed by the graphics engine. Error codes for conditions that the handling routine can expect to be passed by the graphics engine include:

PMERR_INV_IN_AREA
PMERR_INV_IN_PATH
PMERR_DEV_FUNC_NOT_INSTALLED
PMERR_INSUFFICIENT_MEMORY
PMERR_INV_LENGTH_OR_COUNT
PMERR_INV_HDC
PMERR_REALIZE_NOT_SUPPORTED
PMERR_INV_COLOR_FORMAT
PMERR_INV_COLOR_DATA
PMERR_INV_COLOR_START_INDEX
PMERR_INV_COLOR_INDEX.

Remarks

See GpiCreateLogColorTable in the *Programming Reference* for a full description of this function.

GreRealizeColorTable

```
(BOOL) GreRealizeColorTable (hdc, pInstance, lFunction)
```

Causes the system to ensure that, for a realizable color table, the device physical color table is set to the closest possible match to the logical color table.

A device context, such as a printer DC, can implicitly realize a color table when it is created. In this instance, the handling routine need only return Successful. An error must be posted if this function is called for a color table that is cannot be realized.

Support

This function must be supported by the presentation driver.

The sample driver handles this call by bit-map simulation. The call parameters are passed unchanged to the display driver's dispatch table.

Stack Frame

(HDC)	hdc	(device-context handle)
(ULONG)	pInstance	(pointer to instance data)
(ULONG)	lFunction	(High-order word = Flags, Low-order word = NGreRealizeColorTable)

Return Codes

On completion, the handling routine must return BOOLEAN (fSuccess):

TRUE Successful.
FALSE Error.

When an error is detected, the handling routine must call WinSetErrorInfo to post the condition. Error codes for conditions that the handling routine is expected to check include:

PMERR_INV_IN_AREA
PMERR_INV_IN_PATH
PMERR_DEV_FUNC_NOT_INSTALLED
PMERR_INV_HDC
PMERR_REALIZE_NOT_SUPPORTED
PMERR_COL_TABLE_NOT_REALIZABLE.

```
(BOOL) GreUnrealizeColorTable (hdc, pInstance, lFunction)
```

Reverses GreRealizeColorTable by causing the default physical color table for the device to be reinstated.

The logical color table is unaffected by this function.

Support

This function must be supported by the presentation driver.

The sample driver handles this call by bit-map simulation. The call parameters are passed unchanged to the display driver's dispatch table.

Stack Frame

(HDC)	hdc	(device-context handle)
(ULONG)	pInstance	(pointer to instance data)
(ULONG)	lFunction	(High-order word = Flags, Low-order word = NGreUnrealizeColorTable)

Return Codes

On completion, the handling routine must return BOOLEAN (fSuccess):

TRUE Successful.
FALSE Error.

When an error is detected, the handling routine must call WinSetErrorInfo to post the condition. Error codes for conditions that the handling routine is expected to check include:

PMERR_INV_IN_AREA
PMERR_INV_IN_PATH
PMERR_DEV_FUNC_NOT_INSTALLED
PMERR_INV_HDC
PMERR_COL_TABLE_NOT_REALIZED.

GreQueryRealColors

```
(LONG) GreQueryRealColors (hdc, flOptions, iStart, cArray, pArray, pInstance, lFunction)
```

Stores, in the array addressed by pArray, the RGB values of the distinct colors available on the currently associated device.

Support

This function must be supported by the presentation driver.

See module K0COLORS.C of the sample presentation driver.

Stack Frame

(HDC)	hdc	(device-context handle)
(ULONG)	flOptions	(see below)
(LONG)	iStart	(ordinal number of the first color required, see below)
(LONG)	cArray	(number of elements available in the array)
(PLONG)	pArray	(pointer to array in which data is returned)
(ULONG)	pInstance	(pointer to instance data)
(ULONG)	lFunction	(High-order word = Flags, Low-order word = NGreQueryRealColors)

flOptions Valid options are:

LOPT_REALIZED If set, the information is required when the logical color table (if any) is realized.

When this flag is not set, the information is required when the logical color table is not realized.

LOPT_INDEX If set, the handling routine must return the index for each RGB value.

Other flags are reserved and must be zero.

iStart This would normally be zero to start the sequence. This does not necessarily bear any relationship to the color index; the order in which the colors are returned is not defined.

pArray When LOPT_INDEX is set, this is an array of alternating color indexes and values, in the order index1, value1, index2, value2... If there is a color table, colors that are not in the table but available on the device have a special index of CLR_NOINDEX. In RGB mode, the RGB value is returned in the color indexes.

When LOPT_INDEX is not set, this is an array of color values, each value is as defined for CreateLogColorTable on page 12-96.

Return Codes

On completion, the handling routine must return the number of colors returned in the array (cColors) or GPI_ALTERROR if an error occurred.

When an error is detected, the handling routine must call WinSetErrorInfo to post the condition. Error codes for conditions that the handling routine is expected to check include:

PMERR_DEV_FUNC_NOT_INSTALLED
PMERR_INV_HDC
PMERR_INV_LENGTH_OR_COUNT
PMERR_INV_COLOR_OPTIONS
PMERR_INV_COLOR_START_INDEX.

```
(LONG) GreQueryNearestColor (hdc, flOptions, rgbColorin, pInstance, lFunction)
```

Returns the nearest available color to the specified color on the currently associated device even if it is not available in the logical color table; both colors are specified as RGB values.

The color returned is that which is used for drawing primitives, such as lines and text, it does not take account of the possibility of *dithered* colors being used for filling areas. Where dithered colors are used for filling, the color used for text and lines is likely to be different, even when the same color index is selected.

The nearest color is determined by finding its position in *RGB space*. RGB space can be defined as a cube with three axes, representing red, green, and blue color intensities, radiating from one corner, or origin. Moving up the Red axis results in increasing red intensity, and different intensities of cyan can be produced by moving along the Green and Blue axes.

Dithering, for EGA and VGA devices, is done by dividing the RGB space into 9×9×9 cubical cells representing the colors that can be created. You can then determine which cell each RGB color falls into, and make a lookup table to tell which EGA planes to set on or off to create each color.

When this function is called for a monochrome device, the color returned is either the reset color or the contrast color for the device, see "Support for Monochrome Devices" on page 12-93. The sample presentation driver always returns RGB_BLACK, see module K0COLORS.C.

Support

This function must be supported by the presentation driver.

See module K0COLORS.C of the sample presentation driver.

Stack Frame

(HDC)	hdc	(device-context handle)
(ULONG)	flOptions	(see below)
(LONG)	rgbColorIn	(color required)
(ULONG)	pInstance	(pointer to instance data)
(ULONG)	lFunction	(High-order word = Flags, Low-order word = NGreQueryNearestColor)

flOptions The only significant flag is:

LOPT_REALIZED If set, the information is required when the logical color table (if any) is realized, otherwise the information is required when the logical color table is not realized.

Other flags are reserved.

Return Codes

The handling routine must return the nearest available RGB color to that requested (rgbColorOut) or GPI_ALTERROR if an error occurred.

When an error is detected, the handling routine must call WinSetErrorInfo to post the condition. Error codes for conditions that the handling routine is expected to check include:

PMERR_DEV_FUNC_NOT_INSTALLED
PMERR_INV_HDC
PMERR_INV_COLOR_OPTIONS
PMERR_INV_RGBCOLOR.

GreQueryColorIndex

```
(LONG) GreQueryColorIndex (hdc, flOptions, rgbColor, pInstance, lFunction)
```

Returns the logical color index that is closest to the specified RGB color representation for the device; if the color index is RGB mode, the supplied RGB value is returned.

Support

This function must be supported by the presentation driver.

The sample driver handles this call by bit-map simulation. The call parameters are passed unchanged to the display driver's dispatch table.

Stack Frame

(HDC)	hdc	(device-context handle)
(ULONG)	flOptions	(see below)
(LONG)	rgbColor	(color, as an RGB value)
(ULONG)	pInstance	(pointer to instance data)
(ULONG)	lFunction	(High-order word = Flags, Low-order word = NGreQueryColorIndex)

flOptions The only valid option is:

LCOLOPT_REALIZED If set, the information is required when the logical color table (if any) is realized, otherwise the information is required when the logical color table is not realized.

Other flags are reserved and must be zero.

Return Codes

On completion, the handling routine must return the color index providing the closest match to the specified color (lColorIndex) or GPI_ALTERROR if an error occurred.

When an error is detected, the handling routine must call WinSetErrorInfo to post the condition. Error codes for conditions that the handling routine is expected to check include:

PMERR_DEV_FUNC_NOT_INSTALLED
PMERR_INV_HDC
PMERR_INV_COLOR_OPTIONS
PMERR_INV_RGBCOLOR.

```
(LONG) GreQueryRGBColor (hdc, flOptions, iColor, pInstance, lFunction)
```

Returns the actual RGB color that results from the specified color index for the specified device; if the color index is RGB mode, the nearest RGB color (the same as QueryNearestColor) is returned.

All defined indexes are valid, except CLR_DEFAULT, which causes an error to be raised.

Support

This function must be supported by the presentation driver.

The sample driver handles this call by bit-map simulation. The call parameters are passed unchanged to the display driver's dispatch table.

Stack Frame

(HDC)	hdc	(device-context handle)
(ULONG)	flOptions	(see below)
(LONG)	iColor	(color index)
(ULONG)	pInstance	(pointer to instance data)
(ULONG)	lFunction	(High-order word = Flags, Low-order word = NGreQueryRGBColor)

flOptions Valid options are:

LCOLOPT_REALIZED If set, the information is required when the logical color table (if any) is realized, otherwise the information is required when the logical color table is not realized.

LCOLOPT_INDEX When set, the handling routine must return the actual RGB color originally specified for this index. Otherwise, it must return the nearest RGB color for this index, that is, the one which would result from drawing on the specified device.

Other flags are reserved and must be zero.

Return Codes

On completion, the handling routine must return the nearest available RGB color to that requested (rgbColor) or GPI_ALTERROR if an error occurred.

When an error is detected, the handling routine must call WinSetErrorInfo to post the condition. Error codes for conditions that the handling routine is expected to check include:

PMERR_DEV_FUNC_NOT_INSTALLED
PMERR_INV_HDC
PMERR_INV_COLOR_DATA
PMERR_INV_COLOR_OPTIONS
PMERR_INV_COLOR_INDEX.

Query Functions

- GreQueryDeviceBitmaps
- GreQueryDeviceCaps
- GreQueryDevResource
- GreQueryHardcopyCaps.

```
(BOOL) GreQueryDeviceBitmaps (hdc, paOutData, cOutdata, pInstance, lFunction)
```

Stores, in the array addressed by paOutData, a list of bit-map formats supported by the device. The number of formats supported can be found using the GreQueryDeviceCaps function. Each format is returned in a pair of array elements and is in the form (cPlanes, cBitsPerPel). The first pair in the array must be the format that most closely matches the device.

Support

This function must be supported by the presentation driver.

The sample presentation driver handles this call by bit-map simulation. The call parameters are passed unchanged to the display driver's dispatch table.

Stack Frame

(HDC)	hdc	(device-context handle)
(PLONG)	paOutData	(pointer to array where the bit-map format data is returned, excess elements are set to zero)
(LONG)	cOutData	(number of elements in the array pointed to by paOutData, this must be even)
(ULONG)	pInstance	(pointer to instance data)
(ULONG)	lFunction	(High-order word = Flags, Low-order word = NGreQueryDeviceBitmaps)

Return Codes

On completion, the handling routine must return BOOLEAN (fSuccess):

TRUE Successful.
FALSE Error.

When an error is detected, the handling routine must call WinSetErrorInfo to post the condition. Error codes for conditions that the handling routine is expected to check include:

PMERR_DEV_FUNC_NOT_INSTALLED
PMERR_INV_LENGTH_OR_COUNT.

GreQueryDeviceCaps

```
(BOOL) GreQueryDeviceCaps (hdc, iIndex, paOutData, cOutData, pInstance, lFunction)
```

Supports the DevQueryCaps function at the API. The handling routine returns the required information in the device capabilities buffer addressed by paOutData.

Calls to GreQueryDeviceCaps would not usually require the handling routine to return data in all fields in the buffer. The parameters iIndex and cOutData identify the offset to the first field and the count of consecutive field for returned data.

Note: If GreQueryDeviceCaps returns data in the CAPS_DRIVER_VERSION field, the return value should be 00000100H for OS/2 Version 1.1 or 00000120H for OS/2 Version 1.2.

Support

This function must be supported by the presentation driver. See module LO_QUERY.C of the sample presentation driver.

Stack Frame

(HDC)	hdc	(device-context handle)
(LONG)	iIndex	(identifies the first item required, see below)
(PLONG)	paOutData	(pointer to an array where information is returned)
(LONG)	cOutData	(number of items of information to be returned at paOutData)
(ULONG)	pInstance	(pointer to instance data)
(ULONG)	lFunction	(High-order word = Flags, Low-order word = NGreQueryDeviceCaps)

iIndex Indicates the element, within the device capabilities array, at which the presentation driver must begin returning information. (Device capabilities are held by the system in an array of ULONG fields; for details, see DevQueryCaps in the *Programming Reference*.)

Additional information is provided in the array for communication between the presentation driver and the graphics engine: the CAPS_ADDITIONAL_GRAPHICS field has two extra flags, and a CAPS_DEVICE_FONT_SIM field is provided.

The additional flags in CAPS_ADDITIONAL_GRAPHICS are used, in conjunction with the CAPS_FONT_OUTLINE_DEFAULT and CAPS_FONT_IMAGE_DEFAULT flags, by the presentation driver when it wants the graphics engine to manage transforms and mappings for the default fonts that the driver supplies. The flags are:

CAPS_FONT_OUTLINE_MANAGE Set by the driver to indicate that the graphics engine must manage the default outline font.

CAPS_FONT_IMAGE_MANAGE Set by the driver to indicate that the graphics engine must manage the default image font.

Note: If the driver supplies the fonts but wants the engine to manage them, it must pass the font selector to the graphics engine using GreRealizeFont.

The CAPS_DEVICE_FONT_SIM field contains flags that the presentation driver sets if it wants the graphics engine to handle simulations for the default fonts supplied by the driver:

CAPS_DEV_FONT_SIM_BOLD Set by the driver to indicate that the graphics engine should simulate CDEF_BOLD for device fonts.

CAPS_DEV_FONT_SIM_ITALIC Set by the driver to indicate that the graphics engine should simulate CDEF_ITALIC for device fonts.

CAPS_DEV_FONT_SIM_UNDERSCORE Set by the driver to indicate that the graphics engine should simulate CDEF_UNDERSCORE for device fonts.

CAPS_DEV_FONT_SIM_STRIKEOUT Set by the driver to indicate that the graphics engine should simulate CDEF_STRIKEOUT for device fonts.

Note: The font attributes CDEF_... are identified by the cdef.fFlags field in the character-attributes bundle; see page 12-56. In the presentation driver, routines that write character strings should check the cdef.fFlags field to determine whether the call should be passed to the default handling routine in the graphics engine.

Return Codes

On completion, the handling routine must return BOOLEAN (fSuccess):

TRUE Successful.
FALSE Error.

When an error is detected, the handling routine must call WinSetErrorInfo to post the condition. Error codes for conditions that the handling routine is expected to check include:

PMERR_DEV_FUNC_NOT_INSTALLED
PMERR_INV_QUERY_ELEMENT_NO
PMERR_INV_LENGTH_OR._COUNT.

GreQueryDevResource

```
(SHORT) GreQueryDevResource (hdc, lType, id, pInstance, lFunction)
```

Indicates whether a specified resource is available. If the resource is loaded, its handle is returned so that it can be selected into the device context.

These resources, display information, pointers, bit maps, and fonts are stored in .DLL files. Some are linked by the presentation driver when it is first enabled, others are loaded by the application with WinLoadPointer.

The two system fonts are queried by the graphics engine when the presentation driver is loaded. When the presentation driver has its own default font and wants to use it, it returns the handle of the font, as requested. When this function returns a null handle for the system font, the graphics engine default fonts are used instead, see "Font Functions" on page 15-45.

Support

This function must be supported by the presentation driver for display devices. For other devices, the minimum requirement is for the handling routine to return zero to indicate that a requested resource is not available.

Stack Frame

(HDC)	hdc	(device-context handle)
(ULONG)	lType	(resource type, see below)
(ULONG)	id	(defined resource value)
(ULONG)	pInstance	(pointer to instance data)
(ULONG)	lFunction	(High-order word = Flags, Low-order word = NGreQueryDevResource)

lType Resource types returned by the presentation driver are:

RT_DISPLAYINFO

This is a structure containing some of the display constants required by the window manager. This information is required for all display devices that support windows. The format of the DISPLAYINFO structure is:

cb Size of this structure, it is always set to 26.

cxIcon Count of pels for x-width of icon.

cyIcon Count of pels for y-height of icon. When WinLoadPointer is used to load the icon, it is stretched or compressed to the size indicated by cxIcon and cyIcon.

cxPointer Count of pels for x-width of pointer.

cyPointer Count of pels for y-height of pointer. When WinLoadPointer is used to load the pointer, it is stretched or compressed to the size indicated by cxPointer and cyPointer.

cxBorder Count of pels for x-width of horizontal border.

cyBorder Count of pels for y-height vertical border.

cxHSlider Count of pels for x-width of horizontal scroll-bar slider.

cyVSlider Count of pels for y-height of vertical scroll-bar slider.

cxSizeBorder Count of pels for x-width of default border.

cySizeBorder Count of pels for y-height of default border.

cxDeviceAlign Count of pels for horizontal device alignment.

cyDeviceAlign Count of pels for vertical device alignment. Some display devices operate faster when operation coordinates are aligned to a byte, word, or double-word boundary.

These two parameters allow the presentation driver to align windows on these boundaries and so optimize window management operations.

RT_POINTER

Defined system pointers are:

SPTR_ARROW

Left-pointing arrow. This is usually the system default.

SPTR_TEXT
Text-insertion pointer. Typically, this is used when the mouse pointer is on an edit control.

SPTR_WAIT
An hourglass. Used to tell the user to wait while the system is busy.

SPTR_MOVE
Four arrows together, pointing north, south, east, and west. This tells the user that window can be dragged in any of these directions.

SPTR_SIZENWSE
An arrow pointing northwest and southeast. This tells the user that the window can be sized in these directions.

SPTR_SIZENESW
An arrow pointing northeast and southwest. This tells the user that the window can be sized in these directions.

SPTR_SIZEWE
An arrow pointing east and west. This tells the user that the window can be sized in these directions.

SPTR_SIZENS
An arrow pointing north and south. This tells the user that the window can be sized in these directions.

SPTR_APPICON
Usually a blank icon. This is used when a window, that has been sized down to its minimum and has no normal icon, is dragged across the screen.

SPTR_ICONINFORMATION
This pointer is used as part of a message box, when specified in a call to WinMessageBox.

SPTR_ICONQUESTION
This pointer is used as part of a message box, when specified in a call to WinMessageBox.

SPTR_ICONERROR
This pointer is used as part of a message box, when specified in a call to WinMessageBox.

SPTR_ICONWARNING
This pointer is used as part of a message box, when specified in a call to WinMessageBox.

SPTR_ILLEGAL
This pointer is used by the filing system to notify the user of an illegal mouse-directed copy or move operation.

SPTR_MULTFILE
This is used by the file system to indicate a multiple file copy or move operation.

SPTR_PROGRAM
This is used by the file system to indicate a copy or move operation on an executable program file.

SPTR_FILE
This is used by the file system to indicate a copy or move operation on an ordinary file.

SPTR_FOLDER
This is used by the file system to indicate a copy or move operation on an entire directory.

RT_BITMAP
Defined system bit maps are:

SBMP_SYSMENU
The system-menu bit map; that is, the one that is displayed with the top-most level windows. The dimensions of this bit map are the same as those of the 'min' or 'max' button.

This bit map must be drawn with a border on the right-hand side so that when it is drawn, a border is left for the title bar window.

SBMP_SBUPARROW
This is used to display an upward pointing arrow on scroll bars. The height of this bit map is that of the 'min' or 'max' button. Its width is that of the scroll bar. This bit map must be drawn with top and bottom borders. The side borders are provided by the scroll bar.

SBMP_SBDNARROW
This is used to display a downward pointing arrow on scroll bars. The height of this bit map is that of the 'min' or 'max' button. Its width is that of the scroll bar. This bit map must be drawn with top and bottom borders. The side borders are provided by the scroll bar.

SBMP_SBRGARROW
This is used to display a right-pointing arrow on scroll bars. The height of this bit map is calculated from the width of the vertical scroll-bar arrows multiplied by the y-aspect ratio of the display device. Its width is calculated from the height of the vertical scroll bar arrows multiplied by the y-aspect ratio of the device.

This bit map must be drawn with left and right borders. The top and bottom borders are supplied by the scroll bar.

SBMP_SBLFARROW
This is used to display a left-pointing arrow on scroll bars. The height of this bit map is calculated from the width of the vertical scroll-bar arrows multiplied by the y-aspect ratio of the display device. Its width is calculated from the height of the vertical scroll-bar arrows multiplied by the y-aspect ratio of the device.

This bit map must be drawn with left and right borders. The top and bottom borders are supplied by the scroll bar.

SBMP_SBUPDBLARROW
This bit map is used to display an upward-pointing double arrow on vertical scroll bars. The height of this bit map is approximately that of the 'min' or 'max' button. Its width is same as the down scroll-bar double arrow and the up scroll-bar arrow, which is generally narrower than the 'min' button.

This bit map must be drawn with top and bottom borders. The side borders are drawn by the scroll-bar code.

SBMP_SBDNDBLARROW
This bitmap is used to display a downward-pointing double arrow on vertical scroll bars. The height of this bit map is approximately that of the 'min' or 'max' button. Its width is the same as the up scroll-bar double arrow and the up scroll-bar arrow, which is generally narrower than the 'min' button.

This bit map must be drawn with top and bottom borders. The side borders are drawn by the scroll bar code.

SBMP_SBRGDBLARROW
This bit map is used to display a right-pointing double arrow on horizontal scroll bars. The height is calculated from the width of the vertical scroll-bar arrows multiplied by the y-aspect ratio of the display. The width is calculated from the height of the vertical scroll-bar arrows multiplied by the x-aspect ratio of the display.

This bit map must be drawn with a border on the left and right sides. The top and bottom borders are drawn by scroll-bar code.

SBMP_SBLFDBLARROW
This bit map is used to display a left-pointing double arrow on horizontal scroll bars. The height is calculated from the width of the vertical scroll-bar arrows multiplied by the y-aspect ratio of the display. The width is calculated from the height of the vertical scroll-bar arrows multiplied by the x-aspect ratio of the display.

This bit map must be drawn with a border on the left and right sides. The top and bottom borders are drawn by scroll-bar code.

SBMP_MENUCHECK
This bit map is displayed next to a menu item, when the item is checked. Menu items are displayed in the system font and the menu checks are vertically aligned next to them. The height of this bit map must be no greater than the system-font height, to ensure that consecutive menu-check bit maps do not overlap. Its width is arbitrary, but is normally the same as the system-font width.

SBMP_CHECKBOXES
This bit map holds the images of check boxes and radio buttons. The bit map actually contains twelve smaller bit maps in a three by four array. The first row contains four check-box bit maps: unchecked, checked, highlighted unchecked, and highlighted checked. The second row contains similar information for radio buttons and the third contains four halftone check-box bit maps these are: unused, halftone, unused, and highlighted halftone.

The height of the image must be equal or less than the system font height. The width is arbitrary, but must be no more than two average character widths.

SBMP_BTNCORNERS
This contains the rounded corners for pushbuttons. It is arranged as three bit maps, divided into two by two bit-map arrays describing the corners of each bit map. The three bit maps are defined as follows:

1. This contains the corners of an unselected pushbutton which is not a default pushbutton.
2. This holds the corners of a default pushbutton that is currently not selected.
3. This contains the corners of a currently-selected pushbutton, that can be either a default or non-default pushbutton.

SBMP_MINBUTTON
Signifies that a window can be minimized. The height of this bit map is usually the same as the scroll-bar arrow. Its width is about two and a half times the average character width.

This bit map must be drawn with a border on the left-hand side to allow spacing between itself and the title bar.

SBMP_MAXBUTTON
Shows that a window can be maximized. The dimensions of this bit map must be the same as those for the minimize button.

This bit map must be drawn with a border on the left-hand side to allow spacing between itself and the minimize button.

SBMP_RESTOREBUTTON
This replaces the minimize button, when a frame window is minimized, and the maximize button when the window is maximized. Its dimensions must be the same as the maximize and minimize bit maps, and it must be drawn with a border on its left-hand side to allow spacing between itself and the next bit map.

SBMP_DRIVE
Used by the file system to display the logical disk drive.

SBMP_CHILDSYSMENU
System-menu item.

SBMP_FILE
This is used by the file system to indicate an unknown file type.

SBMP_FOLDER
Used by the file system to display a directory.

SBMP_PROGRAM
Used by the file system to mark .EXE and .COM files.

SBMP_TREEPLUS
Used by the file system to indicate there are more subdirectories to view.

SBMP_TREEMINUS
Used by the file system to indicate there are no more subdirectories to view.

SBMP_MENUATTATCHED
Drawn on the right edge of a menu item to indicate that a pulldown menu is attatched to that item.

SBMP_SIZEBOX
Used by some applications to display a 'sizebox' in the bottom-right corner of a frame window.

RT_FONT
The default system fonts are:

SFONT_RASTER
The default image font.

SFONT_OUTLINE.
The default outline font.

Return Codes

This function returns the selector of the indicated resource, zero if no selector is available, or GPI_ALTERROR if an error occurs.

When an error is detected, the handling routine must call WinSetErrorInfo to post the condition. Error codes for conditions that the handling routine is expected to check include:

PMERR_BASE_ERROR
PMERR_DEV_FUNC_NOT_INSTALLED.

```
(LONG) GreQueryHardcopyCaps (hdc, iStart, cCount, pInfo, pInstance, lFunction)
```

Stores information about the hard-copy capabilities of the device in the buffer addressed by pInfo. The information is stored as a sequence of one or more HCINFO structures defining the hard-copy capabilities for one or more form codes.

For drivers that support more than one form code, with the relevant data held in a structure of HCINFO structures, the parameters iStart and cCount identify the starting point in the main structure and the number of HCINFO structures to be stored.

It is usual for this call to be issued twice, initially with a value of zero in cCount, to return the number of forms available. Storage can then be allocated and the function called again with cCount set to the number of forms for which information is required.

Support

This function must be supported by presentation drivers for hard-copy devices. It is not required for display devices.

In the sample presentation driver, this function is supported by a handling routine in module L1_QUERY.C.

Stack Frame

(HDC)	hdc	(device-context handle)
(LONG)	iStart	(index, see below)
(LONG)	cCount	(number of forms, see below)
(PHCINFO)	pInfo	(pointer to buffer for returned form data, see below)
(ULONG)	pInstance	(pointer to instance data)
(ULONG)	lFunction	(High-order word = Flags, Low-order word = NGreQueryHardcopyCaps)

iStart Index to the required starting HCINFO structure. A value of 0 identifies the HCINFO for the first form.

cCount Number of structures to be returned in the buffer. A value of 0 requests the handling routine to set the return code to the number of forms that the driver supports.

pInfo A pointer to the buffer for the returned data. The data is returned as a set of one or more HCINFO structures:

szFormname[32] Character string, containing the name of the form.
cx The width (left to right) in millimeters.
cy The height (top to bottom) in millimeters.
xLeftClip The left clip limit in millimeters.
yBottomClip The bottom clip limit in millimeters.
xRightClip The right clip limit in millimeters.
yTopClip The top clip limit in millimeters.
xPels Number of pels between left and right clip limits.
yPels Number of pels between bottom and top clip limits.
flAttributes Attributes describing the availability of the form:

HCAPS_CURRENT The form is currently available on the device. For devices with multiple paper trays, HCAPS_CURRENT says that the paper required for this form is in the current paper tray.

HCAPS_SELECTABLE The form is installed on the device but has to be selected. That is, the paper tray required for this form is not the current one.

Return Codes

The value returned by the handling routine depends on the initial value of cCount. If cCount=0, return the total number of form codes that the presentation driver supports. For any other value, return the number of HCINFO structures that were transferred to the buffer.

When an error is detected, the handling routine must return DQHC_ERROR and call WinSetErrorInfo to post the condition. Error codes for conditions that the handling routine is expected to check include:

PMERR_INV_LENGTH_OR._COUNT.

GreEscape

The GreEscape handling routine in the presentation driver supports the DevEscape function and its escape codes at the API. Whilst the primary function of GreEscape is to implement the required support for the defined escape codes, it can be used to implement additional escape codes. For additional escape codes, there is a set of defined ranges; the range chosen for the escape code determines how the system will process it when it is received as a parameter to DevEscape.

On entry to GreEscape, the value of lEscape on the stack identifies the escape code. The action taken is determined by the escape code and the physical device that the presentation driver supports.

GreEscape is implemented in module M0ESCAPE.C of the sample presentation driver.

Support

GreEscape with the DEVESC_QUERYESCSUPPORT escape code must be supported by all presentation drivers. Presentation drivers for hard-copy devices should also support the DEVESC_STARTDOC, DEVESC_ABORTDOC, and DEVESC_ENDDOC escape codes. The other escape codes are optional, see Defined Escape Codes below and the text for the individual escape codes on the following pages.

Stack Frame

On entry to the GreEscape routine, the stack frame contains:

(HDC)	hdc	(device-context handle)
(LONG)	lEscape	(escape code)
(LONG)	cInCount	(number of bytes pointed to by pInData)
(PBYTE)	pInData	(pointer to input data structure)
(PLONG)	pcOutCount	(pointer to the number of bytes in output data structure, see below)
(PLONG)	pOutData	(pointer to output data structure)
(ULONG)	pInstance	(pointer to instance data)
(ULONG)	lFunction	(High-order word = Flags, Low-order word = NGreEscape)

pcOutCount If the escape code is one that returns data in the output data structure, the handling routine changes the value addressed by pcOutCount to show the number of bytes returned.

Return Codes

The handling routine should return:

DEV_OK	Successful.
DEVESC_NOTIMPLEMENTED	Escape not implemented for specified code.
DEVESC_ERROR	Error.

When an error is detected, the handling routine must call WinSetErrorInfo to post the condition. Error codes for conditions that the handling routine is expected to check include:

PMERR_DEV_FUNC_NOT_INSTALLED
PMERR_INV_LENGTH_OR_COUNT.

Defined Escape Codes

The following table shows the escape codes that have been defined for Presentation Manager and the devices that they apply to. The GreEscape handling routine should return DEVESC_NOTIMPLEMENTED for escape codes that it does not support.

DEVESC_ABORTDOC	Hard-copy devices
DEVESC_BREAK_EXTRA	PostScript devices only
DEVESC_CHAR_EXTRA	PostScript devices only
DEVESC_DBE_FIRST	DBCS support
DEVESC_DBE_LAST	DBCS support
DEVESC_DRAFTMODE	Hard-copy devices
DEVESC_ENDDOC	Hard-copy devices
DEVESC_FLUSHOUTPUT	Hard-copy devices
DEVESC_GETCP	Hard-copy devices with built in fonts
DEVESC_GETSCALINGFACTOR	Hard-copy devices
DEVESC_NEWFRAME	Hard-copy devices
DEVESC_NEXTBAND	Hard-copy devices that support banding
DEVESC_QUERYESCSUPPORT	All devices (display and hard-copy)
DEVESC_QUERYVIOCELLSIZES	Display devices, see page 12-128
DEVESC_RAWDATA	Hard-copy devices
DEVESC_STARTDOC	Hard-copy devices
DEVESC_STD_JOURNAL	Hard-copy devices.

Ranges for Additional Escape Codes

The following table shows the defined ranges for additional escape codes and shows how the system will process the escape code when it is received as a parameter to DevEscape.

32768 – 40959	Escape is not metafiled or recorded by the spooler. The escape is passed to the presentation driver in all cases.
40960 – 49151	Escape is metafiled but is not recorded by the spooler For an OD_QUEUED device with PM_Q_STD data and for all device types other than OD_METAFILE, the escape is passed to the presentation driver.
49152 – 57343	Escape is metafiled and recorded by the spooler. For an OD_METAFILE device or for OD_QUEUED with PM_Q_STD data, the escape is not passed to the presentation driver.
57344 – 65535	Escape is recorded by the spooler. The escape is passed to the presentation driver except when the DC is an OD_QUEUED device with PM_Q_STD data.

This escape code aborts the current document. The handling routine in the presentation driver discards all data, such as data in a spooler buffer or journal file, received for the current document and closes any files associated with it. The current document is any data back to, and including, the DEVESC_STARTDOC statement.

For a typical implementation of this function, see module M0ESCAPE.C of the sample presentation driver.

Stack Frame

hdc	Device-context handle.
lEscape	DEVESC_ABORTDOC.
cInCount	The handling routine should ignore this parameter.
pInData	The handling routine should ignore this parameter.
pcOutCount	The handling routine should ignore this parameter.
pOutData	The handling routine should ignore this parameter.

Return Codes

The handling routine returns:

DEV_OK	Successful.
DEVESC_NOTIMPLEMENTED	This escape code is not implemented.
DEVESC_ERROR	Error.

GreEscape DEVESC_BREAK_EXTRA

This escape code changes the width of the break character on a PostScript device. The handling routine sets or resets, as determined by the value of clnCount, an extra width value for the break character. Upon completion, the width of the break character is the default width specified by the font plus any extra widths set by DEVESC_BREAK_EXTRA and DEVESC_CHAR_EXTRA; the extra widths can be positive, zero, or negative.

Stack Frame

hdc Device-context handle.

lEscape DEVESC_BREAK_EXTRA.

clnCount Number of bytes pointed to by plnData. When clnCount = 0, the handling routine should reset the extra width to zero.

plnData If clnCount is not equal to zero, plnData is a pointer to a FIXED value. This value is the required extra width defined in world coordinates.

pcOutCount The handling routine should ignore this parameter.

pOutData The handling routine should ignore this parameter.

Return Codes

The handling routine returns:

DEV_OK	Successful.
DEVESC_NOTIMPLEMENTED	This escape code is not implemented.
DEVESC_ERROR	Error.

This escape code changes the width of all characters, including the break character, on a PostScript device. The handling routine sets or resets, as defined by the value of clnCount, an extra width value. Upon completion, the width of a character is the default width specified by the font plus the extra width set by DEVESC_CHAR_EXTRA; the extra width may be positive, zero, or negative.

Stack Frame

hdc	Device-context handle.
lEscape	DEVESC_CHAR_EXTRA.
clnCount	Number of bytes pointed to by plnData. When clnCount = 0, the handling routine should reset the extra width to zero.
plnData	If clnCount is not equal to zero, plnData is a pointer to a FIXED value. This value is the required extra width specified in world coordinates.
pcOutCount	The handling routine should ignore this parameter.
pOutData	The handling routine should ignore this parameter.

Return Codes

The handling routine returns:

DEV_OK	Successful.
DEVESC_NOTIMPLEMENTED	This escape code is not implemented.
DEVESC_ERROR	Error.

GreEscape
DEVESC_DRAFTMODE

This escape code sets draft mode (text mode) on or off. Setting draft mode on tells the presentation driver that the coming page need not contain any graphics or all-points-addressable output; note that this escape code is valid only at a page boundary (for example, after DEVESC_STARTDOC or DEVESC_NEWFRAME).

When draft mode is on, the driver can choose to optimize throughput by:

- Ignoring all graphics primitives such as lines, arcs, and areas
- Using the fonts provided by the output device
- Approximating positions received in calls to functions such as GreSetCurrentPosition and GreCharStringPos to the nearest character position that the output device supports for the current font.

The presentation driver must maintain current attributes such as color, mix, and transforms, when draft mode is on, even though they might have no effect on the draft output. Similarly the driver needs to track font changes and respond by setting the appropriate device font (for example, enlarged, condensed, or italic).

Note: For an OD_QUEUED device with PM_Q_STD data, the spooler records this escape in the buffer and does not pass it to the presentation driver.

For a typical implementation of this function, see module M0ESCAPE.C of the sample presentation driver.

Stack Frame

hdc	Device-context handle.
lEscape	DEVESC_DRAFTMODE.
cInCount	Number of bytes pointed to by pInData, this should be 2.
pInData	Short integer value specifying the mode: 1 for draft mode on, 0 for off.
pcOutCount	The handling routine should ignore this parameter.
pOutData	The handling routine should ignore this parameter.

Return Codes

The handling routine returns:

DEV_OK	Successful.
DEVESC_NOTIMPLEMENTED	This escape code is not implemented.
DEVESC_ERROR	Error.

This escape code ends the current document. The handling routine does whatever work is required to complete the job; for example, a DC for an OD_QUEUED device with PM_Q_STD data would close the spooler buffer and transfer the buffered data to a spool file.

As with DEVESC_STARTDOC, you cannot assume that this escape code will always be issued at the end of a document. When it has not, the DEVESC_ENDDOC work must be done in the BeginCloseDC or CloseDC routine. But note that DEVESC_ENDDOC is mandatory at the API when writing PM_Q_STD data to an OD_QUEUED device.

For a typical implementation of this function, see module M0ESCAPE.C of the sample presentation driver.

Stack Frame

hdc Device-context handle.

lEscape DEVESC_ENDDOC.

cInCount The handling routine should ignore this parameter.

pInData The handling routine should ignore this parameter.

pcOutCount Pointer, on input this specifies the size of the buffer pointed to by pOutData, on output it is set to the number of data bytes returned in this buffer. The input value of this parameter is usually 2. On completion this is set to zero if no ID is returned in pOutData.

pOutData Pointer to a data area in which the jobID of the spooled print job is returned; set to null if there is no jobID, when for example, the printer DC is OD_DIRECT.

Return Codes

The handling routine returns:

DEV_OK	Successful.
DEVESC_NOTIMPLEMENTED	This escape code is not implemented.
DEVESC_ERROR	Error.

GreEscape
DEVESC_FLUSHOUTPUT

This escape code flushes any output received for the current document. The handling routine discards all data, such as data stored in a spooler buffer or journal file, received for the current document.

For a typical implementation of this function, see module M0ESCAPE.C of the sample presentation driver.

Stack Frame

hdc	Device-context handle.
lEscape	DEVESC_FLUSHOUTPUT.
cInCount	The handling routine should ignore this parameter.
pInData	The handling routine should ignore this parameter.
pcOutCount	The handling routine should ignore this parameter.
pOutData	The handling routine should ignore this parameter.

Return Codes

The handling routine returns:

DEV_OK	Successful.
DEVESC_NOTIMPLEMENTED	This escape code is not implemented.
DEVESC_ERROR	Error.

This escape code gets a printer data stream that would set the specified code page in the printer. The data stream should cater for options such as draft and NLQ as defined in the OS2_PM_DRV_DEVMODE dialog.

The handling routine in the presentation driver responds by writing the data stream in the buffer addressed by pOutData and the count of bytes into the LONG value addressed by pcOutCount.

Note: DEVESC_GETCP is not metafiled and is not recorded by the spooler.

Stack Frame

hdc Device-context handle.

lEscape DEVESC_GETCP.

cInCount The handling routine should ignore this parameter.

pInData The handling routine should ignore this parameter.

pcOutCount Pointer to a value that shows the number of bytes in the buffer addressed by pOutData. The handling routine should update this value to show the number of bytes returned in the buffer.

pOutData Pointer to the buffer in which the handling routine returns the required data.

Return Codes

The handling routine returns:

DEV_OK	Successful.
DEVESC_NOTIMPLEMENTED	This escape code is not implemented.
DEVESC_ERROR	Error.

GreEscape
DEVESC_GETSCALINGFACTOR

This escape code gets the incremental scaling factors for the (x,y) axes of a printing device. The handling routine returns the scaling factors as an SFACTORS structure at the location addressed by pOutData and changes the value addressed by pcOutCount to show the number of bytes in the returned structure.

Scaling factors are used for physical devices where one unit on the x axis is not equal to one unit on the y axis. The factors show an arbitrary unit length expressed in x units and y units. The length is chosen so that the number of x units and y units can be expressed as an exponent of 2 and the exponents are returned in the SFACTORS structure. For example, if there are 8 units of x in the arbitrary unit length, lXscale is set to 3.

Note: For an OD_QUEUED device with PM_Q_STD data, the spooler passes this escape to the presentation driver without recording it.

Stack Frame

hdc Device-context handle.

lEscape DEVESC_GETSCALINGFACTOR.

clnCount The handling routine should ignore this parameter.

plnData The handling routine should ignore this parameter.

pcOutCount Number of bytes pointed to by pOutData. On return, this is updated by the handling routine to the number of bytes actually returned.

pOutData Long pointer to SFACTORS structure:

lXscale x scaling factor, as an exponent of two
lYscale y scaling factor, as an exponent of two.

Return Codes

The handling routine returns:

DEV_OK	Successful.
DEVESC_NOTIMPLEMENTED	This escape code is not implemented.
DEVESC_ERROR	Error.

This escape code indicates that there is no more data for the current page.

For a display device, DEVESC_NEWFRAME is similar to GreErasePS but the handling routine should reset the attributes (color and mix). For hard-copy devices, the handling routine would advance the paper to a new page.

Note: For an OD_QUEUED device with PM_Q_STD data, the spooler records this escape in the buffer and does not pass it on to the presentation driver.

For a typical implementation of this function, see module M0ESCAPE.C of the sample presentation driver.

Stack Frame

hdc	Device-context handle.
lEscape	DEVESC_NEWFRAME.
clnCount	The handling routine should ignore this parameter.
plnData	The handling routine should ignore this parameter.
pcOutCount	The handling routine should ignore this parameter.
pOutData	The handling routine should ignore this parameter.

Return Codes

The handling routine returns:

DEV_OK	Successful.
DEVESC_NOTIMPLEMENTED	This escape code is not implemented.
DEVESC_ERROR	Error.

GreEscape
DEVESC_NEXTBAND

This escape code indicates that there is no more data for the current band and gets the coordinates of the next band; if there is no current band, the handling routine returns the coordinates of the first band.

DEVESC_NEXTBAND is used by programs that do their own banding. Drivers for devices that cannot use banded output need not support this escape code. (See "Banding" on page 10-11.)

For a typical implementation of this function, see module M0ESCAPE.C of the sample presentation driver.

Stack Frame

hdc Device-context handle.

lEscape DEVESC_NEXTBAND.

cInCount The handling routine should ignore this parameter.

pInData The handling routine should ignore this parameter.

pcOutCount Number of bytes of data pointed to by pOutData. On return, this is updated to the number of bytes actually returned.

pOutData Long pointer to a BANDRECT structure where the device coordinates of the next band are returned:

xLeft	The x coordinate of the lower-left corner of the rectangular band
yBottom	The y coordinate of the lower left corner of the rectangular band
xRight	The x coordinate of the top right corner of the rectangular band
yTop	The y coordinate of the top right corner of the rectangular band.

An empty rectangle (that is xLeft = xRight, or yTop = yBottom) marks the end of the banding operation.

Return Codes

The handling routine returns:

DEV_OK	Successful.
DEVESC_NOTIMPLEMENTED	This escape code is not implemented.
DEVESC_ERROR	Error.

This escape code queries whether a particular escape code is implemented by the presentation driver. The presentation driver returns DEV_OK if the specified escape is supported.

Note: For an OD_QUEUED device with PM_Q_STD data, the spooler passes this escape on to the presentation driver without recording it.

For a typical implementation of this function, see module M0ESCAPE.C of the sample presentation driver.

Stack Frame

hdc	Device-context handle.
lEscape	DEVESC_QUERYESCSUPPORT.
cInCount	Number of bytes pointed to by pInData.
pInData	Pointer to an escape-code value specifying the escape function to be checked.
pcOutCount	The handling routine should ignore this parameter.
pOutData	The handling routine should ignore this parameter.

Return Codes

The handling routine returns:

DEV_OK	Successful.
DEVESC_NOTIMPLEMENTED	This escape code is not implemented.
DEVESC_ERROR	Error.

GreEscape
DEVESC_QUERYVIOCELLSIZES

This escape code gets details of the VIO cell sizes supported by the presentation driver. The DEVESC_QUERYVIOCELLSIZES escape code must be implemented by presentation drivers for display devices that provide two or more VIO fonts with different cell sizes; other presentation drivers should not implement this escape code.

Stack Frame

hdc Device-context handle.

lEscape DEVESC_QUERYVIOCELLSIZES.

cInCount The handling routine should ignore this parameter.

pInData The handling routine should ignore this parameter.

pcOutCount This is a pointer to the number of bytes pointed to by pOutData. On return, the handling routine updates the value indicated by this pointer to the number of bytes actually returned. This value must be an even multiple of sizeof(LONG).

When the value passed is less than sizeof(LONG), the handling routine must change it to zero. Nothing is loaded at the address indicated by pOutData.

When the value passed is sizeof(LONG), the handling routine must return the number of supported VIO character-cell sizes. The value indicated by pcOutCount is unchanged. The contents of the address indicated by pOutData are updated so that maxcount is the number of VIO cell sizes provided by the device.

When the value passed is greater than sizeof(LONG), the handling routine must update the buffer addressed by pOutData so that:

- maxcount is the number of VIO cell sizes provided by the device.
- count is the number of VIOFONTCELLSIZE structures returned; this can be zero when OutCount is 2*sizeof(LONG).

```
count==((pcOutCount-2*sizeof(LONG))/2*sizeof(LONG))
```

pOutData Pointer to the address of the data returned. The handling routine stores, at this location, the structure:

maxcount Total number of VIO cell sizes provided by the device.
count This is the number of VIOFONTCELLSIZE structures that follow.

Followed by an array of VIOFONTCELLSIZE structures:

xWidth Width of the VIO character cell.
yHeight Height of the VIO character cell.

Return Codes

The handling routine returns:

DEV_OK	Successful.
DEVESC_NOTIMPLEMENTED	This escape code is not implemented.
DEVESC_ERROR	Error.

This escape code sends ASCII data direct to the spooler or device. The action taken by the handling routine is determined by the DC type; for example, and OD_DIRECT DC would send the raw data direct to the kernel device driver.

Note: For an OD_QUEUED device with PM_Q_STD data, the spooler records this call in the buffer and does not pass it on to the presentation driver.

As a general rule, an application should use DEVESC_RAWDATA only for a complete document or frame within a document; DEVESC_RAWDATA must not be mixed with other drawing functions. If DEVESC_RAWDATA and other drawing functions are called in a single frame, the results are dependent on the implementation. For example, the presentation driver might choose to print the raw data and ignore the other drawing calls.

For a typical implementation of this function, see module M0ESCAPE.C of the sample presentation driver.

Stack Frame

hdc	Device-context handle.
lEscape	DEVESC_RAWDATA.
cInCount	Number of bytes pointed to by pInData.
pInData	Pointer to raw data.
pcOutCount	The handling routine should ignore this parameter.
pOutData	The handling routine should ignore this parameter.

Return Codes

The handling routine returns:

DEV_OK	Successful.
DEVESC_NOTIMPLEMENTED	This escape code is not implemented.
DEVESC_ERROR	Error.

GreEscape
DEVESC_STARTDOC

This escape code starts a new document. The handling routine in the presentation driver does whatever initialization it requires to spool or print the document. The driver continues to spool or print data until it receives DEVESC_ENDDOC, this ensures that documents longer than one page are not interspersed with other jobs.

Because DEVESC_STARTDOC is not mandatory at the API, you cannot assume that an application, when it opens a DC for printing, will pass DEVESC_STARTDOC to the presentation driver. To cater for this the presentation driver must initialize a spool file, with no name, or journal file in the CompleteOpenDC handling routine. If this is done, the driver should set a flag so that initialization is not repeated if DEVESC_STARTDOC is received. But note that a handling routine is required for DEVESC_STARTDOC to save the specified document name in the DC instance data.

Note: DEVESC_STARTDOC is mandatory at the API for an OD_QUEUED device with PM_Q_STD data.

When this call is issued for an OD_QUEUED device, the presentation driver must start the recording of data into the spool file by calling SplStdStart. It should also call SplQmStartDoc to pass the name of the spool file to the visual spooler queue manager. See "SplStdStart" on page 17-7 and "SplQmStartDoc" on page 17-17.

For a typical implementation of this function, see module M0ESCAPE.C of the sample presentation driver.

Stack Frame

hdc	Device-context handle.
lEscape	DEVESC_STARTDOC.
cInCount	Number of bytes pointed to by pInData.
pInData	Pointer to a string, containing the name of the document.
pcOutCount	The handling routine should ignore this parameter.
pOutData	The handling routine should ignore this parameter.

Return Codes

The handling routine returns:

DEV_OK	Successful.
DEVESC_NOTIMPLEMENTED	This escape code is not implemented.
DEVESC_ERROR	Error.

This escape code sends a standard journal file to the presentation driver.

Stack Frame

hdc Device-context handle.

lEscape DEVESC_STD_JOURNAL.

cInCount Number of bytes pointed to by pInData.

pInData Pointer to journal data.

pcOutCount The handling routine should ignore this parameter.

pOutData The handling routine should ignore this parameter.

Return Codes

The handling routine returns:

DEV_OK	Successful.
DEVESC_NOTIMPLEMENTED	This escape code is not implemented.
DEVESC_ERROR	Error.

Chapter 13. Mandatory Functions for Display Devices

This chapter describes those functions that must be supported by the display presentation driver for the primary screen.

Although these functions are not required for printer or plotter presentation drivers, such drivers should provide a default routine to handle the functions described in this chapter and should, at FillLogicalDeviceBlock time, put a pointer to the default routine in the relevant entries in the dispatch table. The default routine should post a warning (PMERR_DEV_FUNC_NOT_INSTALLED) and return BOOLEAN TRUE.

The functions described in this chapter are:

- AVIO functions
- Miscellaneous screen functions.

Each description shows what the handling routine is expected to do, the parameters passed to the routine, and the values that the routine should return.

AVIO Functions

- GreCharRect
- GreCharStr
- GreScrollRect
- GreUpdateCursor
- GreDeviceSetAvioFont.

Advanced VIO (AVIO) functions are used to display characters. These functions must be supported for Display DCs, presentation drivers for printer and plotter devices do not support AVIO functions.

When writing to an AVIO presentation space, an application must ensure that windows containing alphanumeric data are device-cell aligned, where appropriate. The presentation driver can determine whether any characters, which are not cell aligned, are visible.

Most column, row, length, width, and height values correspond to cells within the presentation-space logical video buffer (LVB), whose origin is assumed to be at the bottom left-hand corner of the buffer (0,0).

The presentation driver is expected to clip alphanumeric data to the DC region. This is done in the same way as for normal graphics, by enumerating the rectangles, using GreGetClipRects, and clipping each line to a single rectangle.

Although the driver is not expected to test for correlation hits nor to accumulate GPI_BOUNDS, it should accumulate USER_BOUNDS for AVIO functions if the COM_ALT_BOUNDS command flag is set.

The VIO presentation space is passed to the driver as a pointer to a VioPresentationSpace structure. The presentation driver uses this structure to extract the current-state data to allow it to update the display.

The VioPresentationSpace structure is defined as:

PresentationSpaceLock
: Lock. This parameter is not used by the presentation driver.

Sel_LogicalVideoBuffer
: Segment selector for the LVB. The LVB is a two-dimensional array of character cells and is assumed to begin at offset zero within the segment. The driver must not alter the contents of this field.

Sel_BVSControlBlock
: This parameter is not used by the presentation driver.

rgfAVio
: This parameter is not used by the presentation driver.

CellByteSize
: Size in bytes of a cell in the logical video buffer (LVB). This value is either 2 or 4. The driver must not alter the contents of this field.

BufferRowCount
: Number of cell rows in the logical video buffer. The driver must not alter the contents of this field.

BufferColumnCount
: Number of cell columns in the logical video buffer. The driver must not alter the contents of this field.

WindowOriginRow
: Row index for the logical video buffer. This field, together with the WindowOriginColumn (below), indicates the logical video buffer cell that is drawn in the bottom left of the window's client area. The driver must not alter the contents of this field.

WindowOriginColomn
: Column index for logical video buffer, see WindowOriginRow (above). The driver must not alter the contents of this field.

TextCursorRow
: Row coordinate of flashing text cursor, relative to the logical video buffer. The driver must not alter the contents of this field.

TextCursorColumn
: Column coordinate of flashing text cursor, relative to the logical video buffer. The driver must not alter the contents of this field.

TextCursorStartLine
First scan line of a character-cell image, overlaid by the text cursor. Lines in the cell image are numbered from top to bottom, the first line is 0. The driver must not alter the contents of this field.

TextCursorEndLine Last scan line of a character-cell image, overlaid by the text cursor. The driver must not alter the contents of this field.

TextCursorWidth Width of text cursor, in pels. The driver must not alter the contents of this field.

TextCursorVisible Indicates whether the cursor is visible, nonzero, or invisible, zero. The driver must not alter the contents of this field.

CellImageHeight Height of character cell, in pels. When the value passed is zero or invalid, the driver should reset it to the device default value.

CellImageWidth Width of character cell, in pels. When the value passed is zero or invalid, the driver should reset it to the device default value.

CodePageId ID of current code page for this presentation space. When the value passed is zero or invalid, the driver should reset it to the device default value.

WindowHeight This field is not used by the presentation driver.

WindowWidth This field is not used by the presentation driver.

hConsoleDisplayContext
Device-context handle associated with the presentation space. The driver must not alter the contents of this field.

hVioWindow This field is not used by the display driver.

RowOrgLatch See ColOrgLatch, below.

ColOrgLatch Window origin coordinates saved on (WindowOriginRow and WindowOriginColumn) completion of the last call to GreUpdateCursor. RowOrgLatch and ColOrgLatch are used by the display driver to record the state of the currently displayed cursor. They are interrogated by GreUpdateCursor to determine whether the cursor has moved.

CursorRow See CursorCol, below.

CursorCol Cursor coordinates (TextCursorRow and TextCursorColumn) saved on completion of the last call to GreUpdateCursor. These fields are used by the display driver to record the state of the currently displayed cursor so that it can be successfully erased.

CursorStartLine See CursorEndLine, below.

CursorEndLine Cursor start and end lines (TextCursorStartLine and TextCursorEndLine) saved on completion of the last call to GreUpdateCursor. These fields are used by the display driver to record the state of the currently displayed cursor so that it can be successfully erased.

CursorWidth Cursor width (TextCursorWidth) saved on completion of the last call to GreUpdateCursor. This field is used by the display driver to record the state of the currently displayed cursor so that it can be successfully erased.

PartialCellAdjust This is a negative value reflecting the partial cell-height that must be below the bottom of the window to ensure that a complete cell is positioned at the top of the window (actual window height = WindowHeight rounded up to next complete character cell − PartialCellAdjust). The presentation driver must not alter the contents of this field.

XLatch See YLatch, below.

YLatch Pel coodinates of the bottom left corner of the cursor rectangle relative to the bottom left corner of the window.

WidthLatch See HeightLatch, below.

HeightLatch The height and width, in pels, of the cursor saved on completion of the last call to GreUpdateCursor. These are taken to be CellImageHeight and the difference between TextCursorStartLine and TextCursorEndLine, taking account of any wrapping. The width and height latches are used by GreUdateCursor to record the screen region corresponding to an exclusive-OR cursor.

CellHeightLatch The height in pels of the character cell for which the cursor was last drawn, these values were saved on completion of the last call to GreUpdateCursor. This is used to detect cell height changes.

rgfShieldStates Flags:

CursorShowing (0x0001) Cursor is visible on the screen. This flag is maintained by display driver.

fHasTheFocus (0x0002) This window has the input focus. This flag is not modified by display driver.

fCursorIsOn (0x0004) This bit set to indicate that this is the ON phase of the blink cycle. The cursor should be invisible during the OFF phase of the blink cycle. This flag is not modified by display driver.

selfontsloaded[3] Array of selectors for AVIO fonts 1, 2 and 3.

***npmapfontsloaded[3]**
Array of offsets, within the display driver's global-data segment, of the code-page maps for AVIO fonts 1, 2 and 3.

Each character cell is contained in a two-byte, or four-byte, array in the LVB. The format of the character-cell array is:

Code point Position of the character in the code table.

CGA Attribute byte Character attributes. The four low-order bits represent the foreground color and the high-order bits represent the character background color. Each 4-bit color value corresponds to an explicit 24-bit RGB value. The RGB values are defined within the graphics engine and match the colors available on a CGA device.

Extended Attribute byte. This only applies to 4-byte cells. It is defined as:

- **bit 7** underscore.
- **bit 6** reverse video.
- **bit 4** background transparency; when set, the background is transparent, when clear the background is opaque.
- **bit 1-0** character set 0,1,2 or 3.

Spare Attribute byte This only applies to 4-byte character cells. It is reserved for the system.

```
(LONG) GreCharRect (hdc, pps, pCharRect, pInstance, lFunction)
```

Draws a rectangle of character cells from the LVB to the device context. The attributes for each character are applied, by the handling routine, as the character is drawn.

Support

This function must be supported by presentation drivers for display devices.

Stack Frame

(HDC)	hdc	(device-context handle)
(VioPresentationSpace FAR *)	pps	(pointer to the Vio presentation space)
(LPGridRectRef)	pCharRect	(pointer to a block of parameters for the call, see below)
(ULONG)	pInstance	(pointer to instance data)
(ULONG)	lFunction	(High-order word = flags, Low-order word = NGreCharRect)

pCharRect Pointer to a parameter block. This is a GridRectRef structure:

StartRow The starting row, relative to the bottom left of the LVB, of the character rectangle to be drawn.

StartCol The starting column, relative to the bottom left of the LVB, of the character rectangle to be drawn.

RectWidth The width of the rectangle to be updated, in character cells.

RectHeight The height of the rectangle to be updated.

Return Codes

This function returns a LONG value as an error indicator:

NO_ERROR Successful completion.

CE_INVALID_PRESENTATION_SPACE
Error. (For example, invalid CellByteSize.)

Remarks

This call is used to implement the advanced Vio function VioSetOrg.

GreCharStr

```
(LONG) GreCharStr (hdc, pps, pCharStr, pInstance, lFunction)
```

Draws a string of character cells from the LVB to the device context. The attributes for each character are applied, by the handling routine, as the character is drawn.

When the end of a row is reached, the next character is drawn in the first cell of the next row. Character drawing continues until either, the string, or the logical video buffer is exhausted.

Support

This function must be supported by presentation drivers for display devices.

Stack Frame

(HDC)	hdc	(device-context handle)
(VioPresentationSpace FAR *)	pps	(pointer to the Vio presentation space)
(LPGridRectRef)	pCharStr	(pointer to a block of parameters for the call, see below)
(ULONG)	pInstance	(pointer to instance data)
(ULONG)	lFunction	(High-order word = flags, Low-order word = NGreCharStr)

pCharStr Pointer to parameter block. This is a GridStringRef structure:

StartRow The starting row, relative to the bottom left of the LVB, of the character rectangle to be drawn.

StartCol The starting column, relative to the bottom left of the LVB, of the character rectangle to be drawn.

StringLength Number of characters to be written.

Return Codes

This function returns a LONG value as an error indicator:

NO_ERROR Successful completion.
CE_INVALID_ROW_INDEX
CE_INVALID_PRESENTATION_SPACE (For example, invalid CellByteSize.)
GRE_INVALID_COLUMN_INDEX
GRE_NEGATIVE_LENGTH

```
(LONG) GreScrollRect (hdc, pps, paScrollRect, pInstance, lFunction)
```

Scrolls the contents of the LVB through the DC. The contents of the LVB are not affected by this function.

Typically, the presentation driver responds to this call by calling GreCharRect.

An alternative approach is to use is to use the horizontal and vertical movement fields to define a new source rectangle in the DC and to use GreBitblt to transfer the bits. When new information is revealed from the LVB as a result of the scroll, the handling routine calls GreCharRect to update the display. This approach provides considerable performance advantages for devices that support GreBitblt. See "GreBitblt" on page 12-35.

Support

This function must be supported by presentation drivers for display devices.

Stack Frame

(HDC)	hdc	(device-context handle)
(VioPresentationSpace FAR *)	pps	(pointer to the Vio presentation space)
(LPScrollRectRef)	paScrollRect	(pointer to a block of parameters for the call, see below)
(ULONG)	pInstance	(pointer to instance data)
(ULONG)	lFunction	(High-order word = flags, Low-order word = NGreScrollRect)

paScrollRect Pointer to parameter block for this function. This is defined as a ScrollRectRef structure:

StartRow The starting row, relative to the bottom left of the LVB.
StartCol The starting column in the logical video buffer of the character string to be output.
RectWidth The width of the scroll rectangle.
RectDepth The depth of the scroll rectangle.
HorzMovement The number of columns to be scrolled.
VertMovement The number of rows to be scrolled.

Note: Positive values mean movement downward or to the right, negative mean upward or to the left.

lpFillCell Pointer to a cell, containing the character and attributes, to be used for filling the tail of the scroll region. This is only used when GreBitblt is used to implement this function. When this is passed as NULL, the logical video buffer has been updated, the handling routine must, therefore, call GreCharRect to update the display.

Return Codes

This function returns a LONG value as an error indicator:

NO_ERROR Successful completion.
CE_INVALID_PRESENTATION_SPACE Error. (For example, invalid CellByteSize.)

GreUpdateCursor

```
(LONG) GreUpdateCursor (hdc, pps, pInstance, lFunction)
```

This function updates the drawn alphanumeric cursor to match the cursor state information contained in the presentation space. This usually involves removing the previous cursor from the window and drawing the new cursor, if visible, according to the presentation-space information. The new cursor, if visible, is positioned and clipped according to this information and the window's cell-buffer origin and size.

The cursor is drawn as an exclusive-OR bar. Its new position, size, and shape are saved by the handling routine in the Vio presentation space.

Only one cursor may be visible on the screen at any time, and this must be in the window with the input focus; this is enforced, by the system, for Vio functions, but not for AVIO. The AVIO application must alter the visibility of the cursor when changing input focus. When the text cursor collides with AVIO and VIO drawing, the presentation driver must remove and redraw the cursor around the alphanumeric updates.

Note: GreBitblt copies everything including the cursor.

The driver can assume that the values in the RowOrgLatch and CursorWidth fields of the VioPresentationSpace structure parallel the WindowOriginRow and TextCursorWidth, respectively.

Support

This function must be supported by presentation drivers for display devices.

Stack Frame

(HDC)	hdc	(device-context handle)
(VioPresentationSpace FAR *)	pps	(pointer to the Vio presentation space)
(ULONG)	pInstance	(pointer to instance data)
(ULONG)	lFunction	(High-order word = flags, Low-order word = NGreUpdateCursor)

Return Codes

This function returns a LONG value as an error indicator:

NO_ERROR Successful completion.
CE_INVALID_PRESENTATION_SPACE (For example, invalid CellByteSize.)

```
(BOOL) GreDeviceSetAvioFont (hdc, pFontDef, lcid, pInstance, lFunction)
```

Loads, or deletes, an image font used by the AVIO presentation space associated with the device context. When loaded, the font is used by subsequent GreCharRect, GreCharStr, and GreScrollRect calls to draw the character images for the appropriate AVIO set. lLcid identifiers LCID_AVIO_1, LCID_AVIO_2, and LCID_AVIO_3 correspond to AVIO sets 1, 2, and 3 respectively.

If the font is not acceptable for use with an AVIO presentation space, the handling routine returns FALSE to indicate an error. The font must be a fixed-pitch image (raster) font that matches one of the cell sizes for the default font. If the font does not match a supported cell size, the characters are displayed in the presentation space as black cells.

This function can be implemented by storing the font selector and code-page information in the in the selFontsLoaded and npMapFontsLoaded fields of the VioPresentationSpace structure. When this approach is used, the handling routine must get a pointer to the VioPresentationSpace structure so that it can store the information. To do this the handling routine must, call GreGetHandle (page 15-9) with ulIndex = 1 to get the AVIO Presentation Space handle. The handle is then converted to a pointer by calling VioGetPSAddress (page 16-13) through DosCallback. See the *Programming Reference* for information about DosCallBack.

Another approach is for the driver to realize the AVIO fonts required and to store the pointers to the fonts in the DC instance data structure.

Support

This function must be supported by presentation drivers for display devices.

Stack Frame

(HDC)	hdc	(device-context handle)
(PBYTE)	pFontDef	(pointer to a physical font-data structure)
(LONG)	lcid	(local-identifier value of -2, -3, or -4)
(ULONG)	pInstance	(pointer to instance data)
(ULONG)	lFunction	(High-order word = Flags, Low-order word = NGreDeviceSetAVIOFont)

pFontDef A long pointer to a physical font data structure. When the value passed is zero, the handling routine must delete the loaded font corresponding to lLcid. (If the high-order bit of the high-order word in lLcid is set, then the handle is for a device font.)

Return Codes

This function returns BOOLEAN (fSuccess):

TRUE Successful, the font is acceptable for use with an AVIO presentation space.
FALSE Error.

Miscellaneous Screen Functions

- GreSaveScreenBits
- GreRestoreScreenBits
- GreDeviceInvalidateVisRegion
- GreDeviceSetCursor
- GreSetColorCursor
- GreGetPickWindow
- GreSetPickWindow
- GreDeath
- GreResurrection
- GreGetDCOrigin
- GreDeviceSetDCOrigin
- GreQueryDeviceResource.

```
(ULONG) GreSaveScreenBits (hdc, prcRect, pInstance, lFunction)
```

Saves a rectangle of screen bits.

Support

This function must be supported by presentation drivers for display devices. It is permissible to implement this function by returning zero to indicate that the bits were not saved and must, therefore, be saved by the calling routine.

Stack Frame

(HDC)	hdc	(device-context handle)
(PRECTL)	prcRect	(pointer to a screen rectangle defined in screen coordinates)
(ULONG)	pInstance	(pointer to instance data)
(ULONG)	lFunction	(High-order word = flags, Low-order word = NGreSaveScreenBits)

Return Codes

On completion, the handling routine must return a handle to the saved bits (hsbBits) or zero to indicate that the bits were not saved, or that an error occurred.

When an error is detected, the handling routine must call WinSetErrorInfo to post the condition. Error codes for conditions that the handling routine is expected to check include:

PMERR_DEV_FUNC_NOT_INSTALLED
PMERR_INV_HDC.

Remarks

This function lets the user-interface routines improve the performance of dialog boxes.

GreRestoreScreenBits

```
(BOOL) GreRestoreScreenBits (hdc, hsbBits, prcRect, flOptions, pInstance, lFunction)
```

Restores a rectangle of bits to a screen rectangle. This function can also free the handle of the saved bits.

Support

This function must be supported by presentation drivers for display devices.

Stack Frame

(HDC)	hdc	(device-context handle)
(ULONG)	hsbBits	(handle to screen bits to be restored)
(PRECTL)	prcRect	(pointer to a screen rectangle defined in screen coordinates)
(ULONG)	flOptions	(options flags, see below)
(ULONG)	pInstance	(pointer to instance data)
(ULONG)	lFunction	(High-order word = flags, Low-order word = NGreRestoreScreenBits)

flOptions Option flags, valid values are:

RSB_FREE (1) Free the save-bits handle.
RSB_RESTORE (2) Restore the bits to the screen.

Return Codes

On completion, the handling routine must return BOOLEAN (fSuccess):

TRUE Successful.
FALSE Error.

When an error is detected, the handling routine must call WinSetErrorInfo to post the condition. Error codes for conditions that the handling routine is expected to check include:

PMERR_BASE_ERROR
PMERR_INV_IN_AREA
PMERR_INV_IN_PATH
PMERR_HDC_BUSY
PMERR_DEV_FUNC_NOT_INSTALLED
PMERR_COORDINATE_OVERFLOW
PMERR_BITMAP_NOT_SELECTED
PMERR_INV_PRIMITIVE_TYPE
PMERR_INV_SETID
PMERR_INV_CHAR_SET_ATTR
PMERR_INV_MARKER_SET_ATTR
PMERR_INV_PATTERN_SET_ATTR
PMERR_INV_MARKER_SYMBOL_ATTR
PMERR_UNSUPPORTED_ATTR
PMERR_INV_MIX_ATTR
PMERR_INV_BACKGROUND_MIX_ATTR
PMERR_INV_PATTERN_REF_PT_ATTR
PMERR_INV_LINE_WIDTH_ATTR
PMERR_INV_GEOM_LINE_WIDTH_ATTR
PMERR_INV_LINE_TYPE_ATTR
PMERR_INV_LINE_JOIN_ATTR
PMERR_INV_LINE_END_ATTR
PMERR_INV_CHAR_MODE_ATTR
PMERR_INV_CHAR_SHEAR_ATTR
PMERR_INV_CHAR_DIRECTION_ATTR
PMERR_UNSUPPORTED_ATTR_VALUE
PMERR_INSUFFICIENT_MEMORY
PMERR_INV_LENGTH_OR_COUNT
PMERR_INV_CODEPAGE

PMERR_EXCEEDS_MAX_SEG_LENGTH
PMERR_INV_RECT
PMERR_INV_COORDINATE
PMERR_INV_HDC
PMERR_INV_HRGN
PMERR_INV_DC_DATA
PMERR_INCOMPATIBLE_BITMAP
PMERR_INV_REGION_CONTROL
PMERR_INV_DRIVER_NAME
PMERR_INV_BITBLT_STYLE
PMERR_INV_BITBLT_MIX
PMERR_INV_HBITMAP
PMERR_INV_DC_TYPE
PMERR_INV_COORD_SPACE
PMERR_INV_ID
PMERR_BITMAP_IS_SELECTED
PMERR_HBITMAP_BUSY
PMERR_INV_USAGE_PARM
PMERR_INV_INFO_TABLE
PMERR_INV_BITMAP_DIMENSION
PMERR_INCORRECT_DC_TYPE
PMERR_REALIZE_NOT_SUPPORTED
PMERR_INV_COLOR_FORMAT
PMERR_INV_COLOR_DATA
PMERR_INV_COLOR_START_INDEX
PMERR_INV_COLOR_OPTIONS
PMERR_INV_COLOR_INDEX
PMERR_INV_SCAN_START
PMERR_INV_PICK_APERTURE_POSN
PMERR_INV_PATTERN_SET_FONT
PMERR_HUGE_FONTS_NOT_SUPPORTED
PMERR_INV_COLOR_ATTR
PMERR_INV_BACKGROUND_COL_ATTR.

Remarks

Clipping is done, as necessary, on the restored bits.

GreDeviceInvalidateVisRegion

```
(BOOL) GreDeviceInvalidateVisRegion (hdc, cArray, paBlock, pInstance, lFunction)
```

Notifies the presentation driver that the visible region (and DC region) of one or more DCs has changed, and that the affected DCs must revalidate their visible regions before drawing in them. The array identified by paBlock contains a series of structures each of which identifies a DC and supplies the pointer (pInstance) to its instance data. The display driver responds by setting a flag (HDC_IS_DIRTY) in the instance data of each DC identified in array.

The handling routines for all drawing functions should check the HDC_IS_DIRTY flag before drawing. If the flag is set, the routine must call VisRegionCallback (see page 16-14) to revalidate the DC's visible region.

This function allows the system to defer the calculations caused by visible region changes. This enables menus and dialogs to perform more efficiently.

Support

This function must be supported by presentation drivers for display devices.

Stack Frame

(HDC)	hdc	(device-context handle)
(LONG)	cArray	(number of elements in the array)
(PDC_BLOCK)	paBlock	(long pointer to a parameter array, see below)
(ULONG)	pInstance	(pointer to instance data)
(ULONG)	lFunction	(High-order word = flags, Low-order word = NGreDeviceInvalidateVisRegion)

paBlock This is a long pointer to an array of DC_BLOCK structures:

hdc Device-context handle.
pDcl Pointer to instance data.

Return Codes

This function returns BOOLEAN (fSuccess):

TRUE Successful.
FALSE Error.

When an error is detected, the handling routine must call WinSetErrorInfo to post the condition. Error codes for conditions that the handling routine is expected to check include:

PMERR_DEV_FUNC_NOT_INSTALLED.

```
(BOOL) GreDeviceSetCursor (hdc, pptHotspot, hbm, pInstance, lFunction)
```

Sets the cursor bit map that defines the cursor shape.

Support

This function must be supported by presentation drivers for display devices.

Stack Frame

(HDC)	hdc	(device-context handle)
(PPOINTL)	pptHotspot	(pointer to hot-spot coordinates, see below)
(ULONG)	hbm	(bit-map handle used for the cursor image)
(ULONG)	pInstance	(pointer to instance data)
(ULONG)	lFunction	(High-order word = Flags, Low-order word = NGreDeviceSetCursor)

pptHotspot POINTS structure:

x x position of the hot spot within the cursor bit map.
y y position of the hot spot within the cursor bit map.

Return Codes

On completion, the handling routine must return BOOLEAN (fSuccess):

TRUE Successful.
FALSE Error.

When an error is detected, the handling routine must call WinSetErrorInfo to post the condition. Error codes for conditions that the handling routine is expected to check include:

PMERR_DEV_FUNC_NOT_INSTALLED
PMERR_INV_COORDINATE
PMERR_INV_HDC
PMERR_INV_CURSOR_BITMAP.

Remarks

The handling routine takes the previous cursor bit map and replaces it with the one indicated by hbm. If hbm is null, the cursor has no shape and its image is removed from the display screen.

This function is subject to all clipping.

GreSetColorCursor

```
(BOOL) GreSetColorCursor (hdc, pPointerInfo, pInstance, lFunction)
```

Sets the bit maps that define a color cursor or pointer. The handling routine in the presentation driver updates its copy of the pointer definition to that identified by pPointerInfo.

Support

This function must be supported by presentation drivers for display devices.

Stack Frame

(HDC)	hdc	(device-context handle)
(PPOINTERINFO)	pPointerInfo	(pointer to pointer information, see below)
(ULONG)	pInstance	(pointer to instance data)
(ULONG)	lFunction	(High-order word = Flags, Low-order word = NGreDeviceSetCursor)

pPointerInfo Pointer to a POINTERINFO structure. This structure is described in the *C/2 Bindings Reference* for Presentation Manager.

Return Codes

On completion, the handling routine must return BOOLEAN (fSuccess):

TRUE Successful.
FALSE Error.

When an error is detected, the handling routine must call WinSetErrorInfo to post the condition. Error codes for conditions that the handling routine is expected to check include:

PMERR_DEV_FUNC_NOT_INSTALLED
PMERR_INV_COORDINATE
PMERR_INV_HDC
PMERR_INV_CURSOR_BITMAP.

```
(BOOL) GreGetPickWindow (hdc, pPick, pInstance, lFunction)
```

Stores, at the location addressed by pPick, a RECTL structure, giving the position and size of the pick window, in page-coordinate space.

Support

This function must be supported by presentation drivers for display devices.

Stack Frame

(HDC)	hdc	(device-context handle)
(PRECTL)	pPick	(pointer to pick window see below)
(ULONG)	pInstance	(pointer to instance data)
(ULONG)	lFunction	(High-order word = Flags, Low-order word = NGreGetPickWindow)

pPick The pick window is defined as a RECTL structure, in page-coordinate space:

xLeft Minimum x coordinate of window
yBottom Minimum y coordinate
xRight Maximum x coordinate of window
yTop Maximum y coordinate.

Return Codes

On completion, the handling routine must return BOOLEAN (fSuccess):

TRUE Successful.
FALSE Error.

When an error is detected, the handling routine must call WinSetErrorInfo to post the condition. Error codes for conditions that the handling routine is expected to check include:

PMERR_DEV_FUNC_NOT_INSTALLED
PMERR_INV_HDC.

GreSetPickWindow

```
(BOOL) GreSetPickWindow (hdc, pPick, pInstance, lFunction)
```

Sets the position and size of the pick window in page-coordinate space, for subsequent correlation operations.

Support

This function must be supported by presentation drivers for display devices.

Stack Frame

(HDC)	hdc	(device-context handle)
(PRECTL)	pPick	(pointer to pick window, see below)
(ULONG)	pInstance	(pointer to instance data)
(ULONG)	lFunction	(High-order word = Flags, Low-order word = NGreSetPickWindow)

pPick The pick window is defined as a RECTL structure, in page-coordinate space:

xLeft Minimum x coordinate of window
yBottom Minimum y coordinate
xRight Maximum x coordinate of window
yTop Maximum y coordinate.

Return Codes

On completion, the handling routine must return BOOLEAN (fSuccess):

TRUE Successful.
FALSE Error.

When an error is detected, the handling routine must call WinSetErrorInfo to post the condition. Error codes for conditions that the handling routine is expected to check include:

PMERR_INV_IN_AREA
PMERR_INV_IN_PATH
PMERR_HDC_BUSY
PMERR_DEV_FUNC_NOT_INSTALLED
PMERR_COORDINATE_OVERFLOW
PMERR_INV_LENGTH_OR_COUNT
PMERR_INV_HDC
PMERR_INV_COORD_SPACE
PMERR_INV_PICK_APERTURE_POSN.

Remarks

The boundary of the pick window is included in the correlated area.

```
(BOOL) GreDeath (hdc, pInstance, lFunction)
```

Informs the presentation driver that the whole screen is required by another screen group (an application that is not running under Presentation Manager). Any current Presentation Manager applications are set to the background. While this condition exists, the presentation driver behaves exactly as if the screen is present and continues to process all calls, until it needs to access the display adapter interface, it must then wait until the screen is released to resume processing. When GreResurrection is called, a WM_PAINT message prompts the application to redraw the picture.

Support

This function must be supported by presentation drivers for display devices.

Stack Frame

(HDC)	hdc	(device-context handle)
(ULONG)	pInstance	(pointer to instance data)
(ULONG)	lFunction	(High-order word = Flags, Low-order word = NGreDeath)

Return Codes

On completion, the handling routine must return BOOLEAN (fSuccess):

TRUE Successful.
FALSE Error.

When an error is detected, the handling routine must call WinSetErrorInfo to post the condition. Error codes for conditions that the handling routine is expected to check include:

PMERR_DEV_FUNC_NOT_INSTALLED.

Remarks

This function goes directly to the presentation-driver interface (PDI).

GreResurrection

```
(LONG) GreResurection (hdc, cbVmem, pReserved, pInstance, lFunction)
```

This reverses the condition set by GreDeath and restores the screen to the Presentation Manager. Presentation Manager applications are set to the foreground, and the presentation driver is enabled to update the screen for subsequent drawing calls.

Support

This function must be supported by presentation drivers for display devices.

Stack Frame

(HDC)	hdc	(device-context handle)
(LONG)	cbVmem	(number video-memory bytes changed, see below)
(PULONG)	pReserved	(reserved pointer, must be zero)
(ULONG)	pInstance	(pointer to instance data)
(ULONG)	lFunction	(High-order word = Flags, Low-order word = NGreResurrection)

cbVmem This is the number of bytes of video memory that have been corrupted, and is determined by the VIO. The display driver can use this value to determine whether any of its video memory has been destroyed by the application. Some display drivers can ignore this parameter.

Return Codes

On completion, the handling routine must return a long value (lResult):

0 Error.
1 The screen has been successfully redrawn.
2 The screen has not been completely redrawn, further action is required from the application.

When an error is detected, the handling routine must call WinSetErrorInfo to post the condition. Error codes for conditions that the handling routine is expected to check include:

PMERR_DEV_FUNC_NOT_INSTALLED
PMERR_INV_HDC.

Remarks

This function goes directly to the presentation-driver interface (PDI).

```
(BOOL) GreGetDCOrigin ( hdc, pptOrigin, pInstance, lFunction )
```

Queries the origin of the device context.

The DC origin defines the bottom left drawing origin for a display device, or a banded printer device. The DC origin is set by GreSetupDC (page 14-77) at the graphics engine and by GreDeviceSetDCOrigin at the driver ("GreDeviceSetDCOrigin" on page 13-22). The DC origin is stored in the device context instance (DCI) data structure addressed by pInstance.

Support

This function must be supported by presentation drivers for display devices and for hard-copy devices that use banding. The minimum requirement for other devices is for the handling routine to return successful with pptOrigin set to (0,0).

Stack Frame

(HDC)	hdc	(device-context handle)
(PPOINTL)	pptOrigin	(pointer to the (x,y) coordinates of the returned DC origin, in screen coordinates)
(ULONG)	pInstance	(pointer to instance data)
(ULONG)	lFunction	(High-order word = Flags, Low-order word = NGreGetDCOrigin)

Return Codes

This function returns BOOLEAN (fSuccess):

TRUE Successful.
FALSE Error.

GreDeviceSetDCOrigin

```
(BOOL) GreDeviceSetDCOrigin (hdc, pptDC, pInstance, lFunction)
```

Sets the origin of the device context.

A device context, when created, has its origin set to 0,0.

Support

This function must be supported by presentation drivers for display devices and for hard-copy devices that use banding. The minimum requirement for other hard-copy devices is for the handling routine to return TRUE if the origin addressed by pptDC is set to zero, or to log an error and return FALSE.

Stack Frame

(HDC)	hdc	(device-context handle)
(POINTL FAR *)	pptDC	(pointer to the DC origin, see below)
(ULONG)	pInstance	(pointer to instance data)
(ULONG)	lFunction	(High-order word = Flags, Low-order word = NGreDeviceSetDCOrigin)

pptDC This is the offset to the origin of the device context, indicated by hdc. Convert does not add in this offset (see "GreConvert" on page 14-99), the presentation driver must, therefore, add it to all transformed coordinates.

Note: WORLD_COORDINATE to SCREEN_COORDINATE is not a valid conversion.

Return Codes

On completion, the handling routine must return BOOLEAN (fSuccess):

TRUE Successful.
FALSE Error.

When an error is detected, the handling routine must call WinSetErrorInfo to post the condition. Error codes for conditions that the handling routine is expected to check include:

PMERR_DEV_FUNC_NOT_INSTALLED
PMERR_INV_HDC.

Chapter 14. Simulated Functions

These functions are simulated by handling routines in the graphics engine and are called through pointers in the default dispatch table. You may choose to hook any of these functions to improve performance or to exploit special features of the device. Hooking is done by overwriting the pointer in the driver's copy of the dispatch table with a pointer to the driver's own routine; if you do this, save the original pointer in case you need to pass calls to the engine's handling routine.

When the presentation driver has hooked a function, all calls to that function are passed, through the dispatch table, directly to the driver's own handling routine. If the driver decides that it cannot completely handle a hooked function, it can pass the call to the engine's routine for completion.

The functions in this chapter are grouped according to the conditional include sections of the header file PMDDIM.H:

- Arc functions (INCL_GRE_ARCS)
- Path functions (INCL_GRE_PATHS)
- Region functions (INCL_GRE_REGIONS)
- Clip functions (INCL_GRE_CLIP)
- Transform functions (INCL_GRE_XFORMS).

Each description shows what the handling routine is expected to do, the parameters passed to the routine, and the values that the routine should return.

Arc Functions

- GreGetArcParameters
- GreSetArcParameters
- GreArc
- GrePartialArc
- GreFullArcInterior
- GreFullArcBoundary
- GreFullArcBoth
- GreBoxInterior
- GreBoxBoundary
- GreBoxBoth
- GrePolyFillet
- GrePolyFilletSharp
- GrePolySpline.

Drawing functions, such as those listed above, pass individual drawing orders to the graphics engine. The graphics engine then draws, correlates and takes, or takes bounds on the drawing primitives as directed by the flags.

The graphics engine is assumed to clip to the appropriate part of the window, which is the region excluding any window border or frills.

Coordinates are passed as signed 32-bit numbers in a logical space called world-coordinate space.

Angles are passed as signed 32-bit numbers: zero refers to the direction of the positive x axis; 2^{31} represents 360°. Positive values are counterclockwise from the positive x axis.

```
(BOOL) GreGetArcParameters (hdc, pArcParms, pInstance, lFunction)
```

Stores the current arc parameters in the buffer addressed by pArcParms.

Support

This function is supported by the graphics engine and can be hooked by the presentation driver.

Stack Frame

(HDC)	hdc	(device-context handle)
(PARCPARAMS)	pArcParms	(pointer to ARCPARAMS structure)
(ULONG)	pInstance	(pointer to instance data)
(ULONG)	lFunction	(High-order word = Flags, Low-order word = NGreGetArcParameters)

Return Codes

On completion, the handling routine must return BOOLEAN (fSuccess):

TRUE Successful.
FALSE Error.

When an error is detected, the handling routine must call WinSetErrorInfo to post the condition. Error codes for conditions that the handling routine is expected to check include:

PMERR_HDC_BUSY
PMERR_INV_HDC.

GreSetArcParameters

```
(BOOL) GreSetArcParameters (hdc, pArcParms, pInstance, lFunction)
```

Sets the arc parameters.

Support

This function is supported by the graphics engine and can be hooked by the presentation driver.

Stack Frame

(HDC)	hdc	(device-context handle)
(PARCPARAMS)	pArcParms	(pointer to an array containing the arc parameters, see below)
(ULONG)	pInstance	(pointer to instance data)
(ULONG)	lFunction	(High-order word = Flags, Low-order word = NGreSetArcParameters)

pArcParms ARCPARAMS structure:

lP
lQ
lR
lS

See the *Programming Reference* for a full description of the arc parameters structure.

Return Codes

On completion, the handling routine must return BOOLEAN (fSuccess):

TRUE Successful.
FALSE Error.

When an error is detected, the handling routine must call WinSetErrorInfo to post the condition. Error codes for conditions that the handling routine is expected to check include:

PMERR_HDC_BUSY
PMERR_INV_HDC.

```
(LONG) GreArc (hdc, pptPoints, pInstance, lFunction)
```

Draws an arc through three points: the current position and the two points specified in the data structure. Upon completion, the current position is the third point of the arc.

If this function is used within a path or an area definition to continue a figure following a Box or FullArc function, an error (PMERR_INV_NESTED_FIGURES) is posted. This is because GreBox... and GreFullArc... functions generate a closed figure within an area or path definition.

Support

This function is supported by the graphics engine and can be hooked by the presentation driver.

Stack Frame

(HDC)	hdc	(device-context handle)
(PPOINTL)	pptPoints	(pointer to ArcData array, see below)
(ULONG)	pInstance	(pointer to instance data)
(ULONG)	lFunction	(High-order word = Flags, Low-order word = NGreArc)

pptPoints This is a pointer to an array of POINTL structures giving the mid and end points of the arc. If the mid point is coincident with the start or end point, a straight line is drawn from the start point to the end point.

x X-coordinate of point.
y Y-coordinate of point.

Return Codes

cHits

GPI_OK Successful.
GPI_HITS Successful with correlate hit (returned by display drivers when the correlate flag is on and a hit is detected).
GPI_ERROR Error.

When an error is detected, the handling routine must call WinSetErrorInfo to post the condition. Error codes for conditions that the handling routine is expected to check include:

PMERR_BASE_ERROR
PMERR_INV_IN_AREA
PMERR_PATH_LIMIT_EXCEEDED
PMERR_INV_IN_PATH
PMERR_PATH_UNKNOWN
PMERR_HDC_BUSY
PMERR_DEV_FUNC_NOT_INSTALLED
PMERR_COORDINATE_OVERFLOW
PMERR_BITMAP_NOT_SELECTED
PMERR_INV_MATRIX_ELEMENT
PMERR_INV_LENGTH_OR_COUNT
PMERR_INV_RECT
PMERR_INV_HDC
PMERR_INV_COORD_SPACE
PMERR_INV_COLOR_DATA
PMERR_INV_COLOR_INDEX
PMERR_INV_PICK_APERTURE_POSN.

GrePartialArc

```
(LONG) GrePartialArc (hdc, pptCenter, fxMultiplier, fxStart, fxSweep, pInstance, lFunction)
```

Draws a straight line from the current position to the starting point of a partial arc and draws the specified partial arc. Upon completion, the current (x,y) position is the end of the partial arc.

The dimensions of the full arc are defined as a multiplier that is applied to the current arc parameters. The partial arc that is drawn, is the section of the full arc that is enclosed by the specified start and sweep angles.

If this function is used within a path or an area definition to continue a figure following a Box or FullArc function, an error (PMERR_INV_NESTED_FIGURES) is posted. This is because GreBox... and GreFullArc... both generate a closed figure within an area or path definition.

Support

This function is supported by the graphics engine and can be hooked by the presentation driver.

Stack Frame

(HDC)	hdc	(device-context handle)
(PPOINTL)	pptCenter	(pointer to (x,y) coordinates for center of arc)
(FIXED)	fxMultiplier	(Multiplier, see below)
(FIXED)	fxStart	(Start angle, see below)
(FIXED)	fxSweep	(Sweep angle, see below)
(ULONG)	pInstance	(pointer to instance data)
(ULONG)	lFunction	(High-order word = Flags, Low-order word = NGrePartialArc)

fxMultiplier This parameter defines the size of the full arc in relation to an arc drawn with the current arc parameters. The multiplier is a fixed-point value: the high-order word contains the integer portion; the low-order word contains the fractional portion. A value of 64KB gives a multiplier of one. There is an implementation limit of 255 for this value, which must not be negative.

fxStart/fxSweep Start and sweep angles are measured counterclockwise from the x axis at the center of the arc before the arc parameters are applied. If the current arc parameters do not specify a circle, the angles are skewed to the same degree that the full arc is a skewed circle.

The angles are specified as doubleword values in fixed-point format: the high-order word contains the integer portion; the low-order word contains the fractional portion. A value of 6553 gives an angle of 1°.

Both angles must be positive. Whether the arc is drawn in a clockwise or counterclockwise direction is determined by the arc parameters.

An angle greater than 360° is also valid. In this instance, after the initial line, a full arc is drawn, followed by a partial arc of (lSweep MOD 360)°. See also GpiPartialArc in the *Programming Reference*.

Return Codes

On completion, the handling routine must return an integer (cHits) indicating, where appropriate, whether correlation hits were detected:

GPI_OK Successful.

GPI_HITS Successful with correlate hit (returned by display drivers when the correlate flag is on and a hit is detected).

GPI_ERROR Error.

When an error is detected, the handling routine must call WinSetErrorInfo to post the condition. Error codes for conditions that the handling routine is expected to check include:

PMERR_BASE_ERROR
PMERR_INV_IN_AREA
PMERR_PATH_LIMIT_EXCEEDED
PMERR_INV_IN_PATH
PMERR_PATH_UNKNOWN
PMERR_HDC_BUSY
PMERR_DEV_FUNC_NOT_INSTALLED
PMERR_COORDINATE_OVERFLOW
PMERR_BITMAP_NOT_SELECTED
PMERR_INV_LENGTH_OR_COUNT
PMERR_INV_RECT
PMERR_INV_HDC
PMERR_INV_MULTIPLIER
PMERR_INV_ANGLE_PARM
PMERR_INV_COORD_SPACE
PMERR_INV_COLOR_DATA
PMERR_INV_COLOR_INDEX
PMERR_INV_PICK_APERTURE_POSN.

GreFullArcInterior

```
(LONG) GreFullArcInterior (hdc, fxMultiplier, pInstance, lFunction)
```

Draws a filled full arc, using the current pattern attributes, with its center at the current (x,y) position. The dimensions of the arc are defined by a multiplier that is applied to the current arc parameters. GreFullArc... functions do not affect the current position. The arc boundary is not drawn.

When either the COM_PATH or the COM_AREA flag is set this function must raise an error.

When correlating, the handling routine must return a hit when the pick aperture intersects the interior or is completely within the interior, even when the mix is LEAVEALONE, see "Mix Modes" on page 12-50.

Support

This function is supported by the graphics engine and can be hooked by the presentation driver.

Stack Frame

(HDC)	hdc	(device-context handle)
(FIXED)	fxMultiplier	(Multiplier, see below)
(ULONG)	pInstance	(pointer to instance data)
(ULONG)	lFunction	(High-order word = Flags, Low-order word = NGreFullArcInterior)

fxMultiplier This parameter defines the size of the required arc in relation to an arc drawn with the current arc parameters. The multiplier is a fixed-point value: the high-order word contains the integer portion; the low-order word contains the fractional portion. A value of 64KB gives a multiplier of one. The implementation limit of the multiplier is 255. This value must be positive or zero.

Return Codes

On completion, the handling routine must return an integer (cHits) indicating, where appropriate, whether correlation hits were detected:

GPI_OK Successful.
GPI_HITS Successful with correlate hit (returned by display drivers when the correlate flag is on and a hit is detected).
GPI_ERROR Error.

When an error is detected, the handling routine must call WinSetErrorInfo to post the condition. Error codes for conditions that the handling routine is expected to check include:

PMERR_BASE_ERROR
PMERR_INV_IN_AREA
PMERR_NOT_IN_PATH
PMERR_PATH_LIMIT_EXCEEDED
PMERR_INV_IN_PATH
PMERR_PATH_UNKNOWN
PMERR_HDC_BUSY
PMERR_DEV_FUNC_NOT_INSTALLED
PMERR_COORDINATE_OVERFLOW
PMERR_BITMAP_NOT_SELECTED
PMERR_INV_PATTERN_SET_ATTR
PMERR_INV_MIX_ATTR
PMERR_INV_BACKGROUND_MIX_ATTR
PMERR_INV_PATTERN_REF_PT_ATTR
PMERR_INV_LINE_TYPE_ATTR
PMERR_INV_CHAR_MODE_ATTR
PMERR_INV_CHAR_DIRECTION_ATTR
PMERR_INSUFFICIENT_MEMORY
PMERR_INV_LENGTH_OR_COUNT
PMERR_INV_CODEPAGE

PMERR_EXCEEDS_MAX_SEG_LENGTH
PMERR_INV_RECT
PMERR_INV_COORDINATE
PMERR_INV_AREA_CONTROL
PMERR_ALREADY_IN_AREA
PMERR_NOT_IN_AREA
PMERR_INV_HDC
PMERR_INV_NESTED_FIGURES
PMERR_INV_MULTIPLIER
PMERR_INV_HRGN
PMERR_HRGN_BUSY
PMERR_REGION_IS_CLIP_REGION
PMERR_INV_REGION_CONTROL
PMERR_INV_COORD_SPACE
PMERR_INV_COLOR_DATA
PMERR_INV_COLOR_INDEX
PMERR_INV_PICK_APERTURE_POSN
PMERR_INV_PATTERN_SET_FONT
PMERR_HUGE_FONTS_NOT_SUPPORTED
PMERR_INV_COLOR_ATTR
PMERR_INV_BACKGROUND_COL_ATTR.

Remarks

When doing correlation, the handling routine should record a hit when the pick aperture intersects, or is completely within, the interior (even if the mix used for the fill operation is transparent).

GreFullArcBoundary

```
(LONG) GreFullArcBoundary (hdc, fxMultiplier, pInstance, lFunction)
```

Draws a line, using the current line attributes, around the edge of a full arc centered on the current (x,y) position. The dimensions of the arc are defined as a multiplier that is applied to the current arc parameters. GreFullArc... functions do not affect the current position.

When correlating, the handling routine must return a hit when the pick aperture intersects the boundary. See "Mix Modes" on page 12-50.

Support

This function is supported by the graphics engine and can be hooked by the presentation driver.

Stack Frame

(HDC)	hdc	(device-context handle)
(FIXED)	fxMultiplier	(Multiplier, see below)
(ULONG)	pInstance	(pointer to instance data)
(ULONG)	lFunction	(High-order word = Flags, Low-order word = NGreFullArcBoundary)

fxMultiplier The value of this parameter defines the size of the required arc in relation to an arc drawn with the current arc parameters. The multiplier is a fixed-point value: the high-order word contains the integer portion; the low-order word contains the fractional portion. A value of 64KB gives a multiplier of one. The implementation limit of the multiplier is 255. This must not be negative.

Return Codes

On completion, the handling routine must return an integer (cHits) indicating, where appropriate, whether correlation hits were detected:

GPI_OK Successful.
GPI_HITS Successful with correlate hit. (Returned by display drivers when the correlate flag is on and a hit is detected.)
GPI_ERROR Error.

When an error is detected, the handling routine must call WinSetErrorInfo to post the condition. Error codes for conditions that the handling routine is expected to check include:

PMERR_BASE_ERROR
PMERR_INV_IN_AREA
PMERR_NOT_IN_PATH
PMERR_PATH_LIMIT_EXCEEDED
PMERR_INV_IN_PATH
PMERR_PATH_UNKNOWN
PMERR_HDC_BUSY
PMERR_DEV_FUNC_NOT_INSTALLED
PMERR_COORDINATE_OVERFLOW
PMERR_BITMAP_NOT_SELECTED
PMERR_INV_LENGTH_OR_COUNT
PMERR_INV_RECT
PMERR_INV_HDC
PMERR_INV_NESTED_FIGURES
PMERR_INV_MULTIPLIER
PMERR_INV_COORD_SPACE
PMERR_INV_COLOR_DATA
PMERR_INV_COLOR_INDEX
PMERR_INV_PICK_APERTURE_POSN.

Remarks

When doing correlation, the handling routine should record a hit when the pick aperture intersects the boundary line.

GreFullArcBoth

```
(LONG) GreFullArcBoth (hdc, fxMultiplier, pInstance, lFunction)
```

Draws, and fills, a full arc centered on the current (x,y) position; the current line attributes apply to the boundary line, the current pattern attributes to the interior. The dimensions of the arc are defined as a multiplier that is applied to the current arc parameters. This function does not affect the current position. When either the COM_PATH or the COM_AREA flag is set this function must raise an error.

When correlating, the handling routine must return a hit when the pick aperture intersects the boundary or is completely within the interior, even when the mix is LEAVEALONE, see "Mix Modes" on page 12-50.

Support

This function is supported by the graphics engine and can be hooked by the presentation driver.

Stack Frame

(HDC)	hdc	(device-context handle)
(FIXED)	fxMultiplier	(Multiplier, see below)
(ULONG)	pInstance	(pointer to instance data)
(ULONG)	lFunction	(High-order word = Flags, Low-order word = NGreFullArcBoth)

fxMultiplier The value of this parameter defines the size of the required arc in relation to an arc drawn with the current arc parameters. The multiplier is a fixed-point value: the high-order word contains the integer portion; the low-order word contains the fractional portion. A value of 64KB gives a multiplier of one. The implementation limit of the multiplier is 255. This must not be negative.

Return Codes

On completion, the handling routine must return an integer (cHits) indicating, where appropriate, whether correlation hits were detected:

GPI_OK Successful.
GPI_HITS Successful with correlate hit (returned by display drivers when the correlate flag is on and a hit is detected).
GPI_ERROR Error.

When an error is detected, the handling routine must call WinSetErrorInfo to post the condition. Error codes for conditions that the handling routine is expected to check include:

PMERR_BASE_ERROR
PMERR_INV_IN_AREA
PMERR_NOT_IN_PATH
PMERR_PATH_LIMIT_EXCEEDED
PMERR_INV_IN_PATH
PMERR_PATH_UNKNOWN
PMERR_HDC_BUSY
PMERR_DEV_FUNC_NOT_INSTALLED
PMERR_COORDINATE_OVERFLOW
PMERR_BITMAP_NOT_SELECTED
PMERR_INV_PATTERN_SET_ATTR
PMERR_INV_MIX_ATTR
PMERR_INV_BACKGROUND_MIX_ATTR
PMERR_INV_PATTERN_REF_PT_ATTR
PMERR_INV_LINE_TYPE_ATTR
PMERR_INV_CHAR_MODE_ATTR
PMERR_INV_CHAR_DIRECTION_ATTR
PMERR_INSUFFICIENT_MEMORY
PMERR_INV_LENGTH_OR_COUNT
PMERR_INV_CODEPAGE

PMERR_EXCEEDS_MAX_SEG_LENGTH
PMERR_INV_RECT
PMERR_INV_COORDINATE
PMERR_INV_AREA_CONTROL
PMERR_ALREADY_IN_AREA
PMERR_NOT_IN_AREA
PMERR_INV_HDC
PMERR_INV_NESTED_FIGURES
PMERR_INV_MULTIPLIER
PMERR_INV_HRGN
PMERR_HRGN_BUSY
PMERR_REGION_IS_CLIP_REGION
PMERR_INV_REGION_CONTROL
PMERR_INV_COORD_SPACE
PMERR_INV_COLOR_DATA
PMERR_INV_COLOR_INDEX
PMERR_INV_PICK_APERTURE_POSN
PMERR_INV_PATTERN_SET_FONT
PMERR_HUGE_FONTS_NOT_SUPPORTED
PMERR_INV_COLOR_ATTR
PMERR_INV_BACKGROUND_COL_ATTR.

Remarks

When doing correlation, the handling routine should record a hit when the pick aperture intersects the boundary or interior, or is completely within the interior (even if the mix used for the fill operation is LEAVEALONE).

GreBoxInterior

```
(LONG) GreBoxInterior (hdc, pBox, pInstance, lFunction)
```

Draws a rectangular box with one corner at the current (x,y) position, and the opposite corner at the specified (x,y) position. The current (x,y) position does not change.

When this function occurs within an area or path definition, it generates a closed figure. It must not occur within any other figure definition.

Support

This function is supported by the graphics engine and can be hooked by the presentation driver.

Stack Frame

(HDC)	hdc	(device-context handle)
(PPOINTL)	pBox	(pointer to BoxData, see below)
(ULONG)	pInstance	(pointer to instance data)
(ULONG)	lFunction	(High-order word = Flags, Low-order word = NGreBoxInterior)

pBox Pointer to a BoxData structure:

- **ptlOpposite** POINTL structure, defining the opposite corner of the box:
 - **x** x-coordinate of opposite corner
 - **y** y-coordinate of opposite corner.
- **lHRound** Horizontal length of the full axis of an ellipse, used for rounding each corner.
- **lVRound** Vertical length of the full axis of an ellipse, used for rounding each corner.

Return Codes

On completion, the handling routine must return an integer (cHits) indicating, where appropriate, whether correlation hits were detected:

GPI_OK Successful.
GPI_HITS Successful with correlate hit (returned by display drivers when the correlate flag is on and a hit is detected).
GPI_ERROR Error.

When an error is detected, the handling routine must call WinSetErrorInfo to post the condition. Error codes for conditions that the handling routine is expected to check include:

PMERR_BASE_ERROR
PMERR_INV_IN_AREA
PMERR_NOT_IN_PATH
PMERR_PATH_LIMIT_EXCEEDED
PMERR_INV_IN_PATH
PMERR_PATH_UNKNOWN
PMERR_HDC_BUSY
PMERR_DEV_FUNC_NOT_INSTALLED
PMERR_COORDINATE_OVERFLOW
PMERR_BITMAP_NOT_SELECTED
PMERR_INV_PATTERN_SET_ATTR
PMERR_INV_MIX_ATTR
PMERR_INV_BACKGROUND_MIX_ATTR
PMERR_INV_PATTERN_REF_PT_ATTR
PMERR_INV_LINE_TYPE_ATTR
PMERR_INV_CHAR_MODE_ATTR
PMERR_INV_CHAR_DIRECTION_ATTR
PMERR_INSUFFICIENT_MEMORY
PMERR_INV_LENGTH_OR_COUNT
PMERR_INV_CODEPAGE

PMERR_EXCEEDS_MAX_SEG_LENGTH
PMERR_INV_RECT
PMERR_INV_COORDINATE
PMERR_INV_AREA_CONTROL
PMERR_ALREADY_IN_AREA
PMERR_NOT_IN_AREA
PMERR_INV_HDC
PMERR_INV_NESTED_FIGURES
PMERR_INV_BOX_ROUNDING_PARM
PMERR_INV_HRGN
PMERR_HRGN_BUSY
PMERR_REGION_IS_CLIP_REGION
PMERR_INV_REGION_CONTROL
PMERR_INV_COORD_SPACE
PMERR_INV_COLOR_DATA
PMERR_INV_COLOR_INDEX
PMERR_INV_PICK_APERTURE_POSN
PMERR_INV_PATTERN_SET_FONT
PMERR_HUGE_FONTS_NOT_SUPPORTED
PMERR_INV_COLOR_ATTR
PMERR_INV_BACKGROUND_COL_ATTR.

Remarks

The sides of the box (before transformation) are parallel to the x and y axes. The corners of the box can be rounded by means of quarter ellipses of the specified diameters. When the value of either diameter is zero, no rounding occurs. When the value of either diameter exceeds the length of the corresponding side, that length is used as the diameter instead. When the value of the diameters are equal to the value of the sides, the corners are rounded with a quarter circle.

If the current position is (x0,y0), the box is drawn from the current position in a counterclockwise direction. This is significant when, for example, the area mode is BA_WINDING.

When doing correlation, the handling routine should record a hit when the pick aperture intersects, or is completely within, the interior (even if the mix used for the fill operation is LEAVEALONE).

GreBoxBoundary

```
(LONG) GreBoxBoundary (hdc, pBox, pInstance, lFunction)
```

Draws a rectangular box with one corner at the current (x,y) position, and the opposite corner at the specified (x,y) position. The current (x,y) position does not change.

Support

This function is supported by the graphics engine and can be hooked by the presentation driver.

Stack Frame

(HDC)	hdc	(device-context handle)
(PPOINTL)	pBox	(pointer to BoxData, see below)
(ULONG)	pInstance	(pointer to instance data)
(ULONG)	lFunction	(High-order word = Flags, Low-order word = NGreBoxBoundary)

pBox BoxData structure:

- **ptlOpposite** POINTL structure, defining the opposite corner of the box:
 - **x** x-coordinate of opposite corner
 - **y** y-coordinate of opposite corner.
- **lHRound** Horizontal length of the full axis of an ellipse, used for rounding each corner.
- **lVRound** Vertical length of the full axis of an ellipse, used for rounding each corner.

Return Codes

On completion, the handling routine must return an integer (cHits) indicating, where appropriate, whether correlation hits were detected:

GPI_OK Successful.
GPI_HITS Successful with correlate hit (returned by display drivers when the correlate flag is on and a hit is detected).
GPI_ERROR Error.

When an error is detected, the handling routine must call WinSetErrorInfo to post the condition. Error codes for conditions that the handling routine is expected to check include:

PMERR_BASE_ERROR
PMERR_INV_IN_AREA
PMERR_NOT_IN_PATH
PMERR_PATH_LIMIT_EXCEEDED
PMERR_INV_IN_PATH
PMERR_PATH_UNKNOWN
PMERR_HDC_BUSY
PMERR_DEV_FUNC_NOT_INSTALLED
PMERR_COORDINATE_OVERFLOW
PMERR_BITMAP_NOT_SELECTED
PMERR_INV_LENGTH_OR_COUNT
PMERR_INV_RECT
PMERR_INV_HDC
PMERR_INV_NESTED_FIGURES
PMERR_INV_BOX_ROUNDING_PARM
PMERR_INV_COORD_SPACE
PMERR_INV_COLOR_DATA
PMERR_INV_COLOR_INDEX
PMERR_INV_PICK_APERTURE_POSN.

Remarks

The sides of the box (before transformation) are parallel to the x and y axes. The corners of the box can be rounded by means of quarter ellipses of the specified diameters. When the value of either diameter is zero, no rounding occurs. When the value of either diameter exceeds the length of the corresponding side, that length is used as the diameter instead. When the value of the diameters are equal to the value of the sides, the corners are rounded with a quarter circle.

If the current position is (x0,y0), the box is drawn from the current position in a counterclockwise direction.

When doing correlation, the handling routine should record a hit when the pick aperture intersects the boundary.

GreBoxBoth

```
(LONG) GreBoxBoth (hdc, pBox, pInstance, lFunction)
```

Draws, and fills, a rectangular box with one corner at the current (x,y) position, and the opposite corner at the specified (x,y) position. The current (x,y) position does not change.

When this function occurs within an area or path definition, it generates a closed figure. It must not occur within any other figure definition.

Support

This function is supported by the graphics engine and can be hooked by the presentation driver.

Stack Frame

(HDC)	hdc	(device-context handle)
(PPOINTL)	pBox	(pointer to BoxData, see below)
(ULONG)	pInstance	(pointer to instance data)
(ULONG)	lFunction	(High-order word = Flags, Low-order word = NGreBoxBoth)

pBox BoxData structure:

- **ptlOpposite** POINTL structure, defining the opposite corner of the box:
 - **x** x-coordinate of opposite corner
 - **y** y-coordinate of opposite corner.
- **lHRound** Horizontal length of the full axis of an ellipse, used for rounding each corner.
- **lVRound** Vertical length of the full axis of an ellipse, used for rounding each corner.

Return Codes

On completion, the handling routine must return an integer (cHits) indicating, where appropriate, whether correlation hits were detected:

GPI_OK Successful.
GPI_HITS Successful with correlate hit (returned by display drivers when the correlate flag is on and a hit is detected).
GPI_ERROR Error.

When an error is detected, the handling routine must call WinSetErrorInfo to post the condition. Error codes for conditions that the handling routine is expected to check include:

PMERR_BASE_ERROR
PMERR_INV_IN_AREA
PMERR_NOT_IN_PATH
PMERR_PATH_LIMIT_EXCEEDED
PMERR_INV_IN_PATH
PMERR_PATH_UNKNOWN
PMERR_HDC_BUSY
PMERR_DEV_FUNC_NOT_INSTALLED
PMERR_COORDINATE_OVERFLOW
PMERR_BITMAP_NOT_SELECTED
PMERR_INV_PATTERN_SET_ATTR
PMERR_INV_MIX_ATTR
PMERR_INV_BACKGROUND_MIX_ATTR
PMERR_INV_PATTERN_REF_PT_ATTR
PMERR_INV_LINE_TYPE_ATTR
PMERR_INV_CHAR_MODE_ATTR
PMERR_INV_CHAR_DIRECTION_ATTR
PMERR_INSUFFICIENT_MEMORY
PMERR_INV_LENGTH_OR_COUNT
PMERR_INV_CODEPAGE

PMERR_EXCEEDS_MAX_SEG_LENGTH
PMERR_INV_RECT
PMERR_INV_COORDINATE
PMERR_INV_AREA_CONTROL
PMERR_ALREADY_IN_AREA
PMERR_NOT_IN_AREA
PMERR_INV_HDC
PMERR_INV_NESTED_FIGURES
PMERR_INV_BOX_ROUNDING_PARM
PMERR_INV_HRGN
PMERR_HRGN_BUSY
PMERR_REGION_IS_CLIP_REGION
PMERR_INV_REGION_CONTROL
PMERR_INV_COORD_SPACE
PMERR_INV_COLOR_DATA
PMERR_INV_COLOR_INDEX
PMERR_INV_PICK_APERTURE_POSN
PMERR_INV_PATTERN_SET_FONT
PMERR_HUGE_FONTS_NOT_SUPPORTED
PMERR_INV_COLOR_ATTR
PMERR_INV_BACKGROUND_COL_ATTR.

Remarks

The sides of the box (before transformation) are parallel to the x and y axes. The corners of the box can be rounded by means of quarter ellipses of the specified diameters. When the value of either diameter is zero, no rounding occurs. When the value of either diameter exceeds the length of the corresponding side, that length is used as the diameter instead. When the value of the diameters are equal to the value of the sides, the corners are rounded with a quarter circle.

If the current position is (x0,y0), the box is drawn from the current position in a counterclockwise direction.

When doing correlation, the handling routine should record a hit when the pick aperture intersects the boundary or interior, or is completely within the interior (even if the mix used for the fill operation is LEAVEALONE).

GrePolyFillet

```
(LONG) GrePolyFillet (hdc, paptPoint, cPoints, pInstance, lFunction)
```

Draws a fillet on a series of connected lines, with the first line starting at the current (x,y) position.

Upon completion, the current (x,y) position is set to the last point in the series.

Support

This function is supported by the graphics engine and can be hooked by the presentation driver.

Stack Frame

(HDC)	hdc	(device-context handle)
(PPOINTL)	paptPoint	(pointer to Coordinates array, see below)
(LONG)	cPoints	(number of coordinate pairs, if this value is passed as zero, the handling routine does nothing, and returns successful)
(ULONG)	pInstance	(pointer to instance data)
(ULONG)	lFunction	(High-order word = Flags, Low-order word = NGrePolyFillet)

paptPoint An array of cPoints (x,y) pairs that contain the (x,y) coordinates of the end points for the lines.

Return Codes

On completion, the handling routine must return an integer (cHits) indicating, where appropriate, whether correlation hits were detected:

GPI_OK Successful.
GPI_HITS Successful with correlate hit (returned by display drivers when the correlate flag is on and a hit is detected).
GPI_ERROR Error.

When an error is detected, the handling routine must call WinSetErrorInfo to post the condition. Error codes for conditions that the handling routine is expected to check include:

PMERR_BASE_ERROR
PMERR_INV_IN_AREA
PMERR_PATH_LIMIT_EXCEEDED
PMERR_INV_IN_PATH
PMERR_PATH_UNKNOWN
PMERR_HDC_BUSY
PMERR_DEV_FUNC_NOT_INSTALLED
PMERR_COORDINATE_OVERFLOW
PMERR_BITMAP_NOT_SELECTED
PMERR_INV_LENGTH_OR_COUNT
PMERR_INV_RECT
PMERR_INV_HDC
PMERR_INV_COORD_SPACE
PMERR_INV_COLOR_DATA
PMERR_INV_COLOR_INDEX
PMERR_INV_PICK_APERTURE_POSN.

Remarks

The shape of the fillet is controlled by a set of coordinates for a series of two or more connected lines: the fillet is tangential to the start point of the first line and to the end point of the last line. When more than two sets of coordinates are supplied, the fillet passes through the mid points of the intermediate lines.

An individual fillet always lies within the area bounded by the start, end, and control points.

See GpiPolyFillet in the *Programming Reference* for more detail.

```
(LONG) GrePolyFilletSharp (hdc, paptPoint, cPoints, pfxSharp, pInstance, lFunction)
```

Draws a sequence of one or more sharp fillets starting at the current (x,y) position. As each fillet is drawn, the end point for the fillet becomes the start point for the next fillet. Upon completion, the current (x,y) position is the end point of the last fillet.

Each fillet is controlled by a set of coordinates for two connected lines and by a sharpness value. The fillet is tangential to the start point of the first line and to the end point of the second line. The sharpness value is determined from:

Sharpness = WD/DB.
Where:
W is the mid point of the line joining the end points,
B is the control point above the top of the fillet, and
D is the point where the fillet intersects the line WC.

See GpiPolyFillet in the *Programming Reference* for more detail.

Support

This function is supported by the graphics engine and can be hooked by the presentation driver.

Stack Frame

(HDC)	hdc	(device-context handle)
(PPOINTL)	paptPoint	(pointer to Coordinates array, see below)
(LONG)	cPoints	(number of coordinate pairs)
(PFIXED)	pfxSharp	(pointer to Sharpness array, see below)
(ULONG)	pInstance	(pointer to instance data)
(ULONG)	lFunction	(High-order word = Flags, Low-order word = NGrePolyFilletSharp)

paptPoint An array of cPoints (x,y) pairs. These are groups of four elements where each group contains two pairs of (x,y) coordinates for the control and end points of the fillets.

cPoints When this is passed as zero, the function does nothing and returns successful.

pfxSharp An array of sharpness values with one element for each fillet. Sharpness is defined as a fixed-point value: the high-order word contains the integer portion; the low-order word contains the fractional portion. A value of 64KB gives a sharpness of one.

Return Codes

On completion, the handling routine must return an integer (cHits) indicating, where appropriate, whether correlation hits were detected:

GPI_OK Successful.
GPI_HITS Successful with correlate hit (returned by display drivers when the correlate flag is on and a hit is detected).
GPI_ERROR Error.

When an error is detected, the handling routine must call WinSetErrorInfo to post the condition. Error codes for conditions that the handling routine is expected to check include:

PMERR_BASE_ERROR
PMERR_INV_IN_AREA
PMERR_PATH_LIMIT_EXCEEDED
PMERR_INV_IN_PATH
PMERR_PATH_UNKNOWN
PMERR_HDC_BUSY
PMERR_DEV_FUNC_NOT_INSTALLED
PMERR_COORDINATE_OVERFLOW

PMERR_BITMAP_NOT_SELECTED
PMERR_INV_LENGTH_OR_COUNT
PMERR_INV_RECT
PMERR_INV_HDC
PMERR_INV_COORD_SPACE
PMERR_INV_COLOR_DATA
PMERR_INV_COLOR_INDEX
PMERR_INV_PICK_APERTURE_POSN.

```
(LONG) GrePolySpline (hdc, paptPoint, cPoints, pInstance, lFunction)
```

Draws a sequence of one or more Bezier splines starting at the current (x,y) position. As each spline is drawn, the specified end point for the spline becomes the start point for the next spline. Upon completion, the current (x,y) position is the end point of the last spline.

Support

This function is supported by the graphics engine and can be hooked by the presentation driver.

Stack Frame

(HDC)	hdc	(device-context handle)
(PPOINTL)	paptPoint	(pointer to coordinates array, see below)
(LONG)	cPoints	(number of coordinate pairs in the array)
(ULONG)	pInstance	(pointer to instance data)
(ULONG)	lFunction	(High-order word = Flags, Low-order word = NGrePolySpline)

paptPoint An array of (3×cPoints) (x,y) values, in groups of six elements where each group contains three pairs of (x,y) coordinates for the control and end points for the splines.

cPoints When this is passed as zero, the function does nothing and returns successful.

Return Codes

On completion, the handling routine must return an integer (cHits) indicating, where appropriate, whether correlation hits were detected:

GPI_OK Successful.
GPI_HITS Successful with correlate hit (returned by display drivers when the correlate flag is on and a hit is detected).
GPI_ERROR Error.

When an error is detected, the handling routine must call WinSetErrorInfo to post the condition. Error codes for conditions that the handling routine is expected to check include:

PMERR_BASE_ERROR
PMERR_INV_IN_AREA
PMERR_PATH_LIMIT_EXCEEDED
PMERR_INV_IN_PATH
PMERR_PATH_UNKNOWN
PMERR_HDC_BUSY
PMERR_DEV_FUNC_NOT_INSTALLED
PMERR_COORDINATE_OVERFLOW
PMERR_BITMAP_NOT_SELECTED
PMERR_INV_LENGTH_OR_COUNT
PMERR_INV_RECT
PMERR_INV_HDC
PMERR_INV_COORD_SPACE
PMERR_INV_COLOR_DATA
PMERR_INV_COLOR_INDEX
PMERR_INV_PICK_APERTURE_POSN.

Remarks

The shape of each spline is controlled by a set of coordinates for three connected lines: the spline starts at the current position and ends at the end point of the third line. The end points of the first and second lines are used as control points.

An individual spline always lies within the area bounded by the start, end, and control points.

See GpiPolySpline in the *Programming Reference* for more detail.

Area and Path Functions

- GreAreaSetAttributes
- GreBeginArea
- GreEndArea
- GreBeginPath
- GreEndPath
- GreCloseFigure
- GreFillPath
- GreOutlinePath
- GreModifyPath
- GreStrokePath
- GreSelectClipPath
- GreSavePath
- GreRestorePath.

A path is an area, or a shape, that can be used to define:

- Wide lines and curves, to which changes of scale can be applied.
- Shapes and areas for filling.
- Irregular shapes to which subsequent primitives are clipped. This is known as a clip path.

```
(BOOL) GreAreaSetAttributes (hdc, lPrimType, flDefsMask, flAttrsMask, pAttrs, pInstance, lFunction)
```

This function is called by the graphics engine after processing a GreSetAttrs call received inside an area or path bracket. The handling routine in the graphics engine does nothing; its purpose is to provide an entry in the dispatch table that can be hooked by presentation drivers that provide their own area or path simulations.

Note: GreAreaSetAttributes is not supported by OS/2 Version 1.1.

Support

This function must be hooked by drivers that perform their own area or path simulations.

Stack Frame

The parameters passed to GreAreaSetAttributes are identical to those passed to GreSetAttrs:

(HDC)	hdc	(device context handle)
(LONG)	lPrimType	(bundle primitive type, see below)
(ULONG)	flDefsMask	(flags indicating which attributes are to be set to default)
(ULONG)	flAttrsMask	(flags indicating which attributes are to be modified)
(PBUNDLE)	pAttrs	(pointer to the fixed-format bundle record containing the attribute values to be set, see below)
(ULONG)	pInstance	(pointer to instance data)
(ULONG)	lFunction	(High-order word = Flags, Low-order word = NGreAreaSetAttributes)

lPrimType Indicates the bundle type. Valid primitive values are:

PRIM_LINE Pen (line attribute bundle).
PRIM_CHAR Character attribute bundle.
PRIM_MARKER Marker attribute bundle.
PRIM_AREA Pattern attribute bundle.
PRIM_IMAGE Image attribute bundle.

pAttrs This is a long pointer to the fixed-format bundle record containing the attribute values to be set, as specified by flAttrsMask. Only the attribute fields corresponding to attribute flags set in flAttrsMask and not set in flDefsMask, contain valid values. This buffer need only be large enough to contain data for the highest offset attribute referenced.

Return Codes

This function returns BOOLEAN (fSuccess):

TRUE Successful.
FALSE Error.

GreBeginArea

```
(BOOL) GreBeginArea (hdc, flOptions, pInstance, lFunction)
```

Indicates the beginning of a set of drawing functions that define the boundary of an area. All of the boundaries of the area are considered to be part of the interior, and are, therefore, filled. This function has no direct affect on current position, although this can be affected by drawing orders within the boundary definition.

When Box or FullArc functions are used within an area definition, they generate closed figures. They must not be used within another figure definition.

In O/S2 Version 1.2, the maximum number of straight line vertices that describe the area is about 1450.

Support

This function is supported by the graphics engine and can be hooked by the presentation driver.

Stack Frame

(HDC)	hdc	(device-context handle)
(ULONG)	flOptions	(Option flags, see below)
(ULONG)	pInstance	(pointer to instance data)
(ULONG)	lFunction	(High-order word = Flags, Low-order word = NGreBeginArea)

flOptions: These flags determine whether the boundary is to be drawn and the drawing mode.

BA_NOBOUNDARY Do not draw boundary lines.
BA_BOUNDARY Draw boundary lines.
BA_ALTERNATE Alternate mode.
BA_WINDING Winding mode.

The defaults are BA_NOBOUNDARY and BA_ALTERNATE.

Return Codes

The handling routine returns BOOLEAN:

TRUE Successful.
FALSE Error.

Area correlation hits are returned at end area time. No hits are returned for primitives, such as lines and arcs, which form part of the area definition.

When an error is detected, the handling routine must call WinSetErrorInfo to post the condition. Error codes for conditions that the handling routine is expected to check include:

PMERR_BASE_ERROR
PMERR_PATH_LIMIT_EXCEEDED
PMERR_INV_IN_PATH
PMERR_HDC_BUSY
PMERR_DEV_FUNC_NOT_INSTALLED
PMERR_COORDINATE_OVERFLOW
PMERR_INV_LENGTH_OR_COUNT
PMERR_INV_AREA_CONTROL
PMERR_ALREADY_IN_AREA
PMERR_INV_HDC
PMERR_INV_COORD_SPACE.

For more information, see GpiBeginArea in the *Programming Reference*.

Remarks

The following calls are valid when received after GreBeginArea and before GreEndArea:

GreArc

GreAreaSetAttributes (valid only for color, mix, and valid line attributes)

GreBoxBoundary

GreDeviceSetAttributes (valid only for color, mix, and valid line attributes)

GreDeviceSetGlobalAttribute (valid only for foreground color and mix)

GreFullArcBoundary

GrePartialArc

GrePolyFillet

GrePolyFilletSharp

GrePolyLine

GrePolySpline

GreQueryCharStringPos

GreQueryTextBox

GreSetArcParameters

GreSetAttributes (valid only for color, mix, and valid line attributes)

GreSetCurrentPosition

GreSetGlobalAttribute (valid only for foreground color and mix)

GreSetModelXform

plus some internal calls between the presentation driver and the graphics engine.

GreEndArea

```
(LONG) GreEndArea (hdc, flCancel, pInstance, lFunction)
```

Indicates the end of a set of drawing functions that define the boundary of an area. If the final (x,y) position is not the same as the starting position of the last figure in the area, the handling routine must close the figure by drawing a line from the current position to the start of the final figure.

Upon completion, the current (x,y) position is the last (x,y) position specified in the area boundary, unless a closure line was drawn, in which instance the start of the last figure in the area definition becomes the current (x,y) position.

The area fill can be built up in memory or, on devices that have hardware assist for area fill, in the device. For convex figures, there can be a performance gain in simply recording the start and end pel positions across each scan line. Whatever algorithm is used to fill the area, it is crucial that the interior fill is identical in each occurrence. If it is not identical, bit-map operations may fail to join correctly when copied to the screen.

Support

This function is supported by the graphics engine and can be hooked by the presentation driver.

Stack Frame

(HDC)	hdc	(device-context handle)
(ULONG)	flCancel	(Cancel, see below)
(ULONG)	pInstance	(pointer to instance data)
(ULONG)	lFunction	(High-order word = Flags, Low-order word = NGreEndArea)

flCancel The value of this flag specifies whether this call cancels the area definition:

EA_DRAW Draw the area. This is the default.
EA_CANCEL If set, cancel the area definition, otherwise draw the area.

A GreEndArea(Cancel) without a previous GreBeginArea is valid but has no effect. This allows the handling routine to reset an area bracket to a known state when it has no knowledge of the actual current state.

Return Codes

On completion, the handling routine must return an integer (cHits) indicating, where appropriate, whether correlation hits were detected:

GPI_OK Successful.
GPI_HITS Successful with correlate hit (returned by display drivers when the correlate flag is on and a hit is detected, on any part of the interior, regardless of mix.
GPI_ERROR Error.

When an error is detected, the handling routine must call WinSetErrorInfo to post the condition. Error codes for conditions that the handling routine is expected to check include:

PMERR_BASE_ERROR
PMERR_INV_IN_AREA
PMERR_PATH_LIMIT_EXCEEDED
PMERR_INV_IN_PATH
PMERR_HDC_BUSY
PMERR_DEV_FUNC_NOT_INSTALLED
PMERR_COORDINATE_OVERFLOW
PMERR_BITMAP_NOT_SELECTED
PMERR_INV_PATTERN_SET_ATTR
PMERR_INV_MIX_ATTR
PMERR_INV_BACKGROUND_MIX_ATTR
PMERR_INV_PATTERN_REF_PT_ATTR
PMERR_INV_LINE_TYPE_ATTR
PMERR_INV_CHAR_MODE_ATTR

PMERR_INV_CHAR_DIRECTION_ATTR
PMERR_INSUFFICIENT_MEMORY
PMERR_INV_LENGTH_OR_COUNT
PMERR_INV_CODEPAGE
PMERR_EXCEEDS_MAX_SEG_LENGTH
PMERR_INV_RECT
PMERR_INV_COORDINATE
PMERR_INV_AREA_CONTROL
PMERR_NOT_IN_AREA
PMERR_INV_HDC
PMERR_INV_HRGN
PMERR_HRGN_BUSY
PMERR_REGION_IS_CLIP_REGION
PMERR_INV_REGION_CONTROL
PMERR_INV_COORD_SPACE
PMERR_INV_COLOR_DATA
PMERR_INV_COLOR_INDEX
PMERR_INV_PICK_APERTURE_POSN
PMERR_INV_PATTERN_SET_FONT
PMERR_HUGE_FONTS_NOT_SUPPORTED
PMERR_INV_COLOR_ATTR
PMERR_INV_BACKGROUND_COL_ATTR.

GreBeginPath

```
(BOOL) GreBeginPath (hdc, idPath, pInstance, lFunction)
```

Identifies the start of a sequence of figures that define a path.

Character attribute setting functions are not allowed within a path definition.

When Box or FullArc functions are used within a path definition, they generate closed figures. They must not be used within another figure definition.

In O/S2 Version 1.2, the maximum number of straight line vertices that describe the area is about 1450.

For more information, see GpiBeginPath in the *Programming Reference*.

Support

This function is supported by the graphics engine and can be hooked by the presentation driver.

Stack Frame

(HDC)	hdc	(device-context handle)
(LONG)	idPath	(path identifier, this value must be 1)
(ULONG)	pInstance	(pointer to instance data)
(ULONG)	lFunction	(High-order word = Flags, Low-order word = NGreBeginPath)

Return Codes

On completion, the handling routine must return BOOLEAN (fSuccess):

TRUE Successful.
FALSE Error.

When an error is detected, the handling routine must call WinSetErrorInfo to post the condition. Error codes for conditions that the handling routine is expected to check include:

PMERR_BASE_ERROR
PMERR_ALREADY_IN_PATH
PMERR_INV_IN_AREA
PMERR_INV_PATH_ID
PMERR_PATH_LIMIT_EXCEEDED
PMERR_HDC_BUSY
PMERR_DEV_FUNC_NOT_INSTALLED
PMERR_COORDINATE_OVERFLOW
PMERR_INV_LENGTH_OR_COUNT
PMERR_INV_HDC
PMERR_INV_COORD_SPACE.

Remarks

The following calls are valid when received after GreBeginPath and before GreEndPath:

compact

GreArc

GreAreaSetAttributes (valid only for color, mix, and valid line attributes)

GreBoxBoundary

GreCharString (outline characters only)

GreCharStringPos (outline characters only)

GreCloseFigure

GreDeviceSetGlobalAttribute (valid only for foreground color and mix)

GreFullArcBoundary

GrePartialArc

GrePolyFillet

GrePolyFilletSharp

GrePolyLine

GrePolyMarker (outline markers only)

GrePolySpline

GreQueryCharPositions

GreQueryTextBox

GreSetArcParameters

GreSetAttributes (valid only for color, mix, and valid line attributes)

GreSetGlobalAttribute (valid only for foreground color and mix)

GreSetCurrentPosition

GreSetModelXform

plus some internal calls between the presentation driver and the graphics engine.

GreEndPath

```
(BOOL) GreEndPath (hdc, flCancel, pInstance, lFunction)
```

Identifies the end of a sequence of figures that define a path. This function is valid outside a path definition but has no effect. When this occurs, the handling routine should ignore it.

Support

This function is supported by the graphics engine and can be hooked by the presentation driver.

Stack Frame

(HDC)	hdc	(device-context handle)
(ULONG)	flCancel	(Cancel, see below)
(ULONG)	pInstance	(pointer to instance data)
(ULONG)	lFunction	(High-order word = Flags, Low-order word = NGreEndPath)

flCancel The value of flCancel specifies whether this call cancels the path definition:

EA_DRAW Create the current path.
EA_CANCEL Cancel the path definition.

A GreEndPath(Cancel) without a previous GreBeginPath is valid but has no effect. This allows the handling routine to reset a path bracket to a known state when it has no knowledge of the actual current state.

Return Codes

On completion, the handling routine must return BOOLEAN (fSuccess):

TRUE Successful.
FALSE Error.

When an error is detected, the handling routine must call WinSetErrorInfo to post the condition. Error codes for conditions that the handling routine is expected to check include:

PMERR_BASE_ERROR
PMERR_INV_END_PATH_OPTIONS
PMERR_NOT_IN_PATH
PMERR_PATH_LIMIT_EXCEEDED
PMERR_HDC_BUSY
PMERR_DEV_FUNC_NOT_INSTALLED
PMERR_COORDINATE_OVERFLOW
PMERR_INV_HDC.

```
(BOOL) GreCloseFigure (hdc, pInstance, lFunction)
```

Closes a figure within a path definition. This is done by drawing a line from the current (x,y) position to the start point of the figure. Upon completion, the current position is the start point of the figure.

Open figures can be generated by starting a new figure (with a Move function), or by ending the path without first closing the figure.

GreCloseFigure is valid outside of a path definition, when this occurs, it has no effect and the handling routine should ignore it.

For more information, see GpiCloseFigure in the *Programming Reference*.

Support

This function is supported by the graphics engine and can be hooked by the presentation driver.

Stack Frame

(HDC)	hdc	(device-context handle)
(ULONG)	pInstance	(pointer to instance data)
(ULONG)	lFunction	(High-order word = Flags, Low-order word = NGreCloseFigure)

Return Codes

On completion, the handling routine must return BOOLEAN (fSuccess):

TRUE Successful.
FALSE Error.

When an error is detected, the handling routine must call WinSetErrorInfo to post the condition. Error codes for conditions that the handling routine is expected to check include:

PMERR_BASE_ERROR
PMERR_NOT_IN_PATH
PMERR_PATH_LIMIT_EXCEEDED
PMERR_HDC_BUSY
PMERR_DEV_FUNC_NOT_INSTALLED
PMERR_COORDINATE_OVERFLOW
PMERR_INV_LENGTH_OR_COUNT
PMERR_INV_HDC
PMERR_INV_COORD_SPACE.

GreFillPath

```
(LONG) GreFillPath (hdc, idPath, flOptions, pInstance, lFunction)
```

Fills the interior of the closed figure, defined in the path, using the current pattern attributes.

Before filling the path, the handling routine must close any open figures in the path definition. On completion, it must delete the path.

All of the boundaries of the area are considered to be part of the interior, and are, therefore, filled.

Support

This function is supported by the graphics engine and can be hooked by the presentation driver.

Stack Frame

(HDC)	hdc	(device-context handle)
(LONG)	idPath	(path ID, must be 1)
(ULONG)	flOptions	(OptionFlags, see below)
(ULONG)	pInstance	(pointer to instance data)
(ULONG)	lFunction	(High-order word = Flags, Low-order word = NGreFillPath)

flOptions These flags determine how the path is to be filled.

FPATH_ALTERNATE	Fill is done using the odd/even (alternate) rule
FPATH_WINDING	Fill is done using the winding rule.

Return Codes

On completion, the handling routine must return an integer (cHits) indicating, where appropriate, whether correlation hits were detected:

GPI_OK	Successful.
GPI_HITS	Successful with correlate hit (returned by display drivers when the correlate flag is on and a hit is detected).
GPI_ERROR	Error.

When an error is detected, the handling routine must call WinSetErrorInfo to post the condition. Error codes for conditions that the handling routine is expected to check include:

PMERR_BASE_ERROR
PMERR_INV_IN_AREA
PMERR_INV_PATH_ID
PMERR_PATH_LIMIT_EXCEEDED
PMERR_INV_FILL_PATH_OPTIONS
PMERR_INV_IN_PATH
PMERR_PATH_UNKNOWN
PMERR_HDC_BUSY
PMERR_DEV_FUNC_NOT_INSTALLED
PMERR_COORDINATE_OVERFLOW
PMERR_BITMAP_NOT_SELECTED
PMERR_INSUFFICIENT_MEMORY
PMERR_INV_LENGTH_OR_COUNT
PMERR_INV_RECT
PMERR_INV_COORDINATE
PMERR_INV_HDC
PMERR_INV_HRGN
PMERR_HRGN_BUSY
PMERR_REGION_IS_CLIP_REGION
PMERR_INV_REGION_CONTROL

PMERR_INV_COORD_SPACE
PMERR_INV_COLOR_DATA
PMERR_INV_COLOR_INDEX
PMERR_INV_PICK_APERTURE_POSN.

GreOutlinePath

```
(LONG) GreOutlinePath (hdc, idPath, flOptions, pInstance, lFunction)
```

Draws the boundary of the path indicated by idPath. This function is used to draw area boundaries.

Support

This function is supported by the graphics engine and can be hooked by the presentation driver.

Stack Frame

(HDC)	hdc	(device-context handle)
(LONG)	idPath	(path ID, must be 1)
(ULONG)	flOptions	(all these flags are reserved)
(ULONG)	pInstance	(pointer to instance data)
(ULONG)	lFunction	(High-order word = Flags, Low-order word = NGreOutlinePath)

Return Codes

On completion, the handling routine must return an integer (cHits) indicating, where appropriate, whether correlation hits were detected:

GPI_OK Successful.
GPI_HITS Successful with correlate hit (returned by display drivers when the correlate flag is on and a hit is detected).
GPI_ERROR Error.

When an error is detected, the handling routine must call WinSetErrorInfo to post the condition. Error codes for conditions that the handling routine is expected to check include:

PMERR_BASE_ERROR
PMERR_INV_IN_AREA
PMERR_INV_PATH_ID
PMERR_INV_FILL_PATH_OPTIONS
PMERR_INV_IN_PATH
PMERR_PATH_UNKNOWN
PMERR_HDC_BUSY
PMERR_DEV_FUNC_NOT_INSTALLED
PMERR_COORDINATE_OVERFLOW
PMERR_BITMAP_NOT_SELECTED
PMERR_INV_LENGTH_OR_COUNT
PMERR_INV_RECT
PMERR_INV_HDC
PMERR_INV_COORD_SPACE
PMERR_INV_COLOR_DATA
PMERR_INV_COLOR_INDEX
PMERR_INV_PICK_APERTURE_POSN.

```
(BOOL) GreModifyPath (hdc, idPath, cmdMode, pInstance, lFunction)
```

Modifies a path. When the transform is singular, this function is invalid and causes an error to be logged.

Support

This function is supported by the graphics engine and can be hooked by the presentation driver.

Stack Frame

(HDC)	hdc	(device-context handle)
(LONG)	idPath	(path ID)
(LONG)	cmdMode	(Modify mode, see below)
(ULONG)	pInstance	(pointer to instance data)
(ULONG)	lFunction	(High-order word = Flags, Low-order word = NGreModifyPath)

cmdMode The only valid flag is:

MPATH_STROKE This replaces the path with one enclosing the shape produced by stroking the original path using the current geometric wide line attribute. Any open figures within the path are not closed.

The envelope includes the affects of line joins, and line ends, according to the current values of these attributes.

Notes:

1. Where, for example, a line joins an arc, the common point is handled according to the line-join attribute in the current line attribute bundle, see "Line (Pen) Attributes" on page 12-52.
2. When a figure is closed using GreCloseFigure, the line-join attribute in the current line bundle is applied to the start and end points.
3. The envelope takes account of any crossings. For example, a stroked X does not have a hole in the middle if it is subsequently drawn in exclusive-OR mode.

Return Codes

On completion, the handling routine must return BOOLEAN (fSuccess):

TRUE Successful.
FALSE Error.

When an error is detected, the handling routine must call WinSetErrorInfo to post the condition. Error codes for conditions that the handling routine is expected to check include:

PMERR_BASE_ERROR
PMERR_INV_IN_AREA
PMERR_INV_PATH_ID
PMERR_PATH_LIMIT_EXCEEDED
PMERR_INV_FILL_PATH_OPTIONS
PMERR_INV_IN_PATH
PMERR_PATH_UNKNOWN
PMERR_INV_MODIFY_PATH_MODE
PMERR_HDC_BUSY
PMERR_DEV_FUNC_NOT_INSTALLED
PMERR_COORDINATE_OVERFLOW
PMERR_BITMAP_NOT_SELECTED
PMERR_INV_MATRIX_ELEMENT
PMERR_INV_PATTERN_REF_PT_ATTR
PMERR_INSUFFICIENT_MEMORY

PMERR_INV_LENGTH_OR_COUNT
PMERR_INV_RECT
PMERR_INV_COORDINATE
PMERR_INV_HDC
PMERR_INV_HRGN
PMERR_HRGN_BUSY
PMERR_INV_TRANSFORM_TYPE
PMERR_REGION_IS_CLIP_REGION
PMERR_INV_REGION_CONTROL
PMERR_INV_COORD_SPACE
PMERR_INV_COLOR_DATA
PMERR_INV_COLOR_INDEX
PMERR_INV_PICK_APERTURE_POSN.

Remarks

When this function has been done on a path, the only subsequent operations allowed are:

- GreFillPath, in winding mode.
- GreSelectClipPath, in winding mode.

```
(LONG) GreStrokePath (hdc, idPath, flOptions, pInstance, lFunction)
```

This function converts a path to the envelope of a wide line, using the current geometric line attribute (see "Line (Pen) Attributes" on page 12-52). The converted path is filled using winding mode and the current area attributes (see "Area (Pattern) Attributes" on page 12-54), and then drawn. When the path has been drawn it is deleted. This function is exactly equivalent to GreModifyPath followed by GreFillPath, it is provided to allow the presentation driver to optimize its storage.

Note: GreModifyPath and GreStrokePath are the only functions that can cause geometric wide lines to be constructed.

Support

This function is supported by the graphics engine and can be hooked by the presentation driver.

Stack Frame

(HDC)	hdc	(device-context handle)
(LONG)	idPath	(path ID, the only valid value is 1)
(ULONG)	flOptions	(reserved, must be zero)
(ULONG)	pInstance	(pointer to instance data)
(ULONG)	lFunction	(High-order word = Flags, Low-order word = NGreStrokePath)

Return Codes

On completion, the handling routine must return an integer (cHits) indicating, where appropriate, whether correlation hits were detected:

GPI_OK Successful.
GPI_HITS Successful with correlate hit (returned by display drivers when the correlate flag is on and a hit is detected).
GPI_ERROR Error.

When an error is detected, the handling routine must call WinSetErrorInfo to post the condition. Error codes for conditions that the handling routine is expected to check include:

PMERR_BASE_ERROR
PMERR_INV_IN_AREA
PMERR_INV_PATH_ID
PMERR_PATH_LIMIT_EXCEEDED
PMERR_INV_FILL_PATH_OPTIONS
PMERR_INV_IN_PATH
PMERR_PATH_UNKNOWN
PMERR_INV_MODIFY_PATH_MODE
PMERR_HDC_BUSY
PMERR_DEV_FUNC_NOT_INSTALLED
PMERR_COORDINATE_OVERFLOW
PMERR_BITMAP_NOT_SELECTED
PMERR_INV_MATRIX_ELEMENT
PMERR_INV_PATTERN_REF_PT_ATTR
PMERR_INSUFFICIENT_MEMORY
PMERR_INV_LENGTH_OR_COUNT
PMERR_INV_RECT
PMERR_INV_COORDINATE
PMERR_INV_HDC
PMERR_INV_HRGN
PMERR_HRGN_BUSY
PMERR_INV_TRANSFORM_TYPE
PMERR_REGION_IS_CLIP_REGION
PMERR_INV_REGION_CONTROL

PMERR_INV_COORD_SPACE
PMERR_INV_COLOR_DATA
PMERR_INV_COLOR_INDEX
PMERR_INV_PICK_APERTURE_POSN.

```
(BOOL) GreSelectClipPath (hdc, idPath, flOptions, pInstance, lFunction)
```

Resets or modifies the currently selected clip path.

Before modifying the clip path, the handling routine must close all open figures in the path. The modified clip path consists of the areas common to both the old clip path and the figures in the specified path. The modified clip path is bound in device coordinates by the constituent paths when they were defined and is used for all subsequent drawing, including filling.

All of the boundaries of the area are considered to be part of the interior and are not clipped.

Upon completion, the handling routine deletes the old clip path, indicated by idPath, allowing a new path definition, which can coexist with the new clip path, to begin as required.

For more information, see GpiSetClipPath in the *Programming Reference*.

Support

This function is supported by the graphics engine and can be hooked by the presentation driver.

Stack Frame

(HDC)	hdc	(device-context handle)
(LONG)	idPath	(path ID, see below)
(ULONG)	flOptions	(option flags, see below)
(ULONG)	pInstance	(pointer to instance data)
(ULONG)	lFunction	(High-order word = Flags, Low-order word = NGreSelectClipPath)

idPath When this is passed as zero, the path is reset to no clipping. Otherwise, the new clip path is intersected with the existing clip path.

flOptions The following flags determine the path mode:

SCP_ALTERNATE Use alternate mode for intersection.
SCP_WINDING Use winding mode for intersection.

The default is SCP_ALTERNATE.

The following flags determine whether the new path should replace or intersect the initial path:

SCP_AND Intersect the two clip paths. If idPath is not 1, this flag is invalid and the handling routine must log an error.
SCP_RESET Replace the initial clip path. If idPath is not 0, this flag is invalid and the handling routine must log an error.

The default is SCP_RESET.

Other Flags are reserved and must be 0.

Return Codes

On completion, the handling routine must return BOOLEAN (fSuccess):

TRUE Successful.
FALSE Error.

When an error is detected, the handling routine must call WinSetErrorInfo to post the condition. Error codes for conditions that the handling routine is expected to check include:

PMERR_BASE_ERROR
PMERR_INV_IN_AREA
PMERR_INV_PATH_ID
PMERR_PATH_LIMIT_EXCEEDED
PMERR_INV_IN_PATH
PMERR_PATH_UNKNOWN

PMERR_INV_CLIP_PATH_OPTIONS
PMERR_HDC_BUSY
PMERR_DEV_FUNC_NOT_INSTALLED
PMERR_COORDINATE_OVERFLOW
PMERR_BITMAP_NOT_SELECTED
PMERR_INSUFFICIENT_MEMORY
PMERR_INV_LENGTH_OR_COUNT
PMERR_INV_RECT
PMERR_INV_COORDINATE
PMERR_INV_HDC
PMERR_INV_HRGN
PMERR_HRGN_BUSY
PMERR_REGION_IS_CLIP_REGION
PMERR_INV_REGION_CONTROL
PMERR_INV_COORD_SPACE
PMERR_INV_COLOR_DATA
PMERR_INV_COLOR_INDEX
PMERR_INV_PICK_APERTURE_POSN.

```
(BOOL) GreSavePath (hdc, cSave, pInstance, lFunction)
```

This function is called during SaveDC and OpenDC to allow simulations to save their local data structures.

When a new DC is created, this function is called with a count of 1 to initialize its local data.

A Save with a count of one more than the current, generates intervening levels.

Support

This function is supported by the graphics engine and can be hooked by the presentation driver.

Stack Frame

(HDC)	hdc	(device-context handle)
(LONG)	cSave	(DC save level, this must not be a negative value)
(ULONG)	pInstance	(pointer to instance data)
(ULONG)	lFunction	(High-order word = Flags, Low-order word = NGreSavePath)

Return Codes

On completion, the handling routine must return BOOLEAN (fSuccess):

TRUE Successful.
FALSE Error.

When an error is detected, the handling routine must call WinSetErrorInfo to post the condition. Error codes for conditions that the handling routine is expected to check include:

PMERR_HDC_BUSY
PMERR_INV_HDC.

Remarks

This function is required in case an installed simulation is in use.

GreRestorePath

```
(BOOL) GreRestorePath (hdc, cSave, pInstance, lFunction)
```

This function is called during RestoreDC and CloseDC to allow simulations to restore their local data structures.

When a DC is closed, this function is called with a count of zero to free its local data.

Support

This function is supported by the graphics engine and can be hooked by the presentation driver.

Stack Frame

(HDC)	hdc	(device-context handle)
(LONG)	cSave	(DC save level, see below)
(ULONG)	pInstance	(pointer to instance data)
(ULONG)	lFunction	(High-order word = Flags, Low-order word = NGreRestorePath)

cSave Indicates the saved DC state that the handling routine should use to restore the path.

When cSave is passed as −1, the handling routine resets the path to its initial state (−1 is passed to this routine from GreResetDC and is the only valid negative value). A value of zero also indicates that the path should be retored to its initial state; other positive values identify which saved level should be restored (see "Enable subfunction 08H − RestoreDCState" on page 11-16).

Return Codes

On completion, the handling routine must return BOOLEAN (fSuccess):

TRUE Successful.
FALSE Error.

When an error is detected, the handling routine must call WinSetErrorInfo to post the condition. Error codes for conditions that the handling routine is expected to check include:

PMERR_HDC_BUSY
PMERR_INV_HDC.

Remarks

This function is required in case an installed simulation is in use.

Region Functions

- GreGetRegionBox
- GreGetRegionRects
- GreOffsetRegion
- GrePtInRegion
- GreRectInRegion
- GreCreateRectRegion
- GreDestroyRegion
- GreSetRectRegion
- GreCombineRegion
- GreCombineRectRegion
- GreCombineShortLineRegion
- GreEqualRegion
- GrePaintRegion
- GreSetRegionOwner.

Regions are defined as rectangles in device coordinates. They are inclusive at the bottom left boundary and exclusive at the top right boundary. That is, the top-right coordinates are outside the rectangle they define and the bottom-left coordinates are inside the rectangle. When both coordinate pairs are equal, the rectangle dimension is zero.

GreGetRegionBox

```
(LONG) GreGetRegionBox (hdc, hrgn, prcRect, pInstance, lFunction)
```

Loads the buffer pointed to by prcRect with the dimensions of the tightest rectangle round the region indicated by hrgn. This function raises an error when hrgn is the handle of the currently selected clip region.

Support

This function is supported by the graphics engine and can be hooked by the presentation driver.

Stack Frame

(HDC)	hdc	(device-context handle)
(HRGN)	hrgn	(region handle)
(PRECTL)	prcRect	(pointer to rectangle, in device coordinates, see below)
(ULONG)	pInstance	(pointer to instance data)
(ULONG)	lFunction	(High-order word = Flags, Low-order word = NGreGetRegionBox)

prcRect RECTL structure:

xLeft	Minimum x coordinate of rectangle
yBottom	Minimum y coordinate
xRight	Maximum x coordinate of rectangle
yTop	Maximum y coordinate.

Return Codes

On completion, the handling routine returns an integer (lComplexity), indicating the complexity of the region:

RGN_ERROR	Error
RGN_NULL	Null region
RGN_RECT	RECTangular region
RGN_COMPLEX	COMPLEX region (more than 1 rectangle).

When an error is detected, the handling routine must call WinSetErrorInfo to post the condition. Error codes for conditions that the handling routine is expected to check include:

PMERR_INV_HRGN
PMERR_HRGN_BUSY
PMERR_REGION_IS_CLIP_REGION.

Remarks

When the region is empty, the rectangle is returned with the right boundary equal to the left and the top boundary equal to the bottom.

This function must return an error when the region indicated by hrgn is selected into the DC as a clip region.

```
(BOOL) GreGetRegionRects (hdc, hrgn, prcBoundRect, pControl, parcRect, pInstance, lFunction)
```

Fills the array pointed to by parcRect with a list of (x,y) coordinate pairs specifying the rectangles that, together, define the region associated with hrgn. All rectangle coordinates are inclusive at the bottom-left corner and exclusive at the top right.

This function raises an error when hrgn is the handle of a currently selected clip region.

Support

This function is supported by the graphics engine and can be hooked by the presentation driver.

Stack Frame

(HDC)	hdc	(device-context handle)
(HRGN)	hrgn	(region handle)
(PRECTL)	prcBoundRect	(pointer to bounding rectangle, see below)
(PRGNRECT)	pControl	(pointer to control structure, see below)
(PRECTL)	parcRect	(pointer to array of rectangle structures that define the region)
(ULONG)	pInstance	(pointer to instance data)
(ULONG)	lFunction	(High-order word = Flags, Low-order word = NGreGetRegionRects)

prcBoundRect RECTL structure:

xLeft Minimum x coordinate of window
yBottom Minimum y coordinate
xRight Maximum x coordinate of window
yTop Maximum y coordinate.

Only rectangles intersecting this bounding rectangle are returned. If this pointer is NULL, all rectangles in the region are enumerated. If this pointer is not NULL, only rectangles that intersect the bounding rectangle are returned, each of these rectangles is the intersection of the bounding rectangle with a rectangle in the region.

pControl This is a pointer to a RGNRECT structure, consisting of four USHORT parameters:

lrcStart The rectangle number to start enumerating (one means start at the beginning).
crc The number of rectangles that can fit into the buffer (at least 1).
crcReturned Number of rectangles that were written into the buffer (if less than lrcStart, there are no more rectangles to enumerate).
usDirection The direction in which the rectangles are listed:

RECTDIR_LFRT_TOPBOT Left to right, top to bottom
RECTDIR_RTLF_TOPBOT Right to left, top to bottom
RECTDIR_LFRT_BOTTOP Left to right, bottom to top
RECTDIR_RTLF_BOTTOP Right to left, bottom to top.

Return Codes

On completion, the handling routine must return BOOLEAN (fSuccess):

TRUE Successful.
FALSE Error.

When an error is detected, the handling routine must call WinSetErrorInfo to post the condition. Error codes for conditions that the handling routine is expected to check include:

PMERR_INV_RECT
PMERR_INV_COORDINATE
PMERR_INV_HRGN
PMERR_HRGN_BUSY
PMERR_REGION_IS_CLIP_REGION
PMERR_INV_REGION_CONTROL.

```
(BOOL) GreOffsetRegion (hdc, hrgn, pdpt, pInstance, lFunction)
```

Moves the given region by the specified amounts (unless the region is in use as a clipping region, when an error is raised).

Support

This function is supported by the graphics engine and can be hooked by the presentation driver.

Stack Frame

(HDC)	hdc	(device-context handle)
(HRGN)	hrgn	(region handle)
(PPOINTL)	pdpt	(pointer to offset by which region is to be moved, see below)
(ULONG)	pInstance	(pointer to instance data)
(ULONG)	lFunction	(High-order word = Flags, Low-order word = NGreOffsetRegion)

pdpt Offsets, in device coordinates:

dx
dy.

Return Codes

On completion, the handling routine must return BOOLEAN (fSuccess):

TRUE Successful.
FALSE Error.

When an error is detected, the handling routine must call WinSetErrorInfo to post the condition. Error codes for conditions that the handling routine is expected to check include:

PMERR_COORDINATE_OVERFLOW
PMERR_INV_COORDINATE
PMERR_INV_HRGN
PMERR_HRGN_BUSY
PMERR_REGION_IS_CLIP_REGION.

GrePtInRegion

```
(LONG) GrePtInRegion (hdc, hrgn, ppt, pInstance, lFunction)
```

Checks whether a point lies within a region.

This function raises an error when hrgn is the handle of the currently selected clip region.

Support

This function is supported by the graphics engine and can be hooked by the presentation driver.

Stack Frame

(HDC)	hdc	(device-context handle)
(HRGN)	hrgn	(region handle)
(PPOINTL)	ppt	(pointer to (x,y) point specified in device coordinates)
(ULONG)	pInstance	(pointer to instance data)
(ULONG)	lFunction	(High-order word = Flags, Low-order word = NGrePtInRegion)

Return Codes

On completion, the handling routine returns an integer (lInside) indicating whether the point is inside the region:

RGN_ERROR Error
PRGN_OUTSIDE Not in region
PRGN_INSIDE In region.

When an error is detected, the handling routine must call WinSetErrorInfo to post the condition. Error codes for conditions that the handling routine is expected to check include:

PMERR_INV_RECT
PMERR_INV_COORDINATE
PMERR_INV_HRGN
PMERR_HRGN_BUSY
PMERR_REGION_IS_CLIP_REGION.

```
(LONG) GreRectInRegion (hdc, hrgn, prcRect, pInstance, lFunction)
```

Checks whether any part of a given rectangle lies within the specified region.

GreRectInRegion raises an error if hrgn is the handle of the currently selected clip region.

Support

This function is supported by the graphics engine and can be hooked by the presentation driver.

Stack Frame

(HDC)	hdc	(device-context handle)
(HRGN)	hrgn	(region handle)
(PRECTL)	prcRect	(pointer to rectangle, in device coordinates, see below)
(ULONG)	pInstance	(pointer to instance data)
(ULONG)	lFunction	(High-order word = Flags, Low-order word = NGreRectInRegion)

prcRect RECTL structure:

xLeft Minimum x coordinate of rectangle
yBottom Minimum y coordinate
xRight Maximum x coordinate of rectangle
yTop Maximum y coordinate.

Return Codes

On completion, the handling routine returns an integer (lInside) indicating whether the rectangle is inside the region:

RRGN_ERROR Error
RRGN_OUTSIDE Not in region
RRGN_PARTIAL Partially in region
RRGN_INSIDE All in region.

When an error is detected, the handling routine must call WinSetErrorInfo to post the condition. Error codes for conditions that the handling routine is expected to check include:

PMERR_INV_RECT
PMERR_INV_COORDINATE
PMERR_INV_HRGN
PMERR_HRGN_BUSY
PMERR_REGION_IS_CLIP_REGION.

GreCreateRectRegion

```
(HRGN) GreCreateRectRegion (hdc, paRegion, cRect, pInstance, lFunction)
```

Creates a region by taking the OR of a series of rectangles. When no rectangles are specified (cRect is zero), an empty region is created.

Support

This function is supported by the graphics engine and can be hooked by the presentation driver.

Stack Frame

(HDC)	hdc	(device-context handle)
(PRECT)	paRegion	(pointer to region definition, an array of rectangle structures, see below)
(LONG)	cRect	(number of rectangles in the region definition, if this is zero, an empty region is created)
(ULONG)	pInstance	(pointer to instance data)
(ULONG)	lFunction	(High-order word = Flags, Low-order word = NGreCreateRectRegion)

paRegion This is a pointer to an array of rectangles which define the region. Each rectangle is described by a RECTL structure:

xLeft Minimum x coordinate of rectangle
yBottom Minimum y coordinate
xRight Maximum x coordinate of rectangle
yTop Maximum y coordinate.

For each rectangle, xRight must be equal to or greater than xLeft, and yTop must be equal to or greater than yBottom.

The bottom and left boundaries of each rectangle are part of the interior of the region. The top and right boundaries are not.

Return Codes

On completion, the handling routine returns the region handle (hrgn) or RGN_ERROR if an error occurred.

When an error is detected, the handling routine must call WinSetErrorInfo to post the condition. Error codes for conditions that the handling routine is expected to check include:

PMERR_INSUFFICIENT_MEMORY
PMERR_INV_LENGTH_OR_COUNT
PMERR_INV_RECT
PMERR_INV_COORDINATE
PMERR_INV_HRGN.

```
(BOOL) GreDestroyRegion (hdc, hrgn, pInstance, lFunction)
```

Destroys the specified region, unless it has been selected as a clipping region, in which instance an error is raised. When a null region is specified, GreDestroyRegion does not delete any region and it returns without logging an error.

Support

This function is supported by the graphics engine and can be hooked by the presentation driver.

Stack Frame

(HDC)	hdc	(device-context handle)
(HRGN)	hrgn	(Region handle)
(ULONG)	pInstance	(pointer to instance data)
(ULONG)	lFunction	(High-order word = Flags, Low-order word = NGreDestroyRegion)

Return Codes

On completion, the handling routine must return BOOLEAN (fSuccess):

TRUE Successful.
FALSE Error.

When an error is detected, the handling routine must call WinSetErrorInfo to post the condition. Error codes for conditions that the handling routine is expected to check include:

PMERR_INV_HRGN
PMERR_HRGN_BUSY
PMERR_REGION_IS_CLIP_REGION.

When an error occurs, the region is not deleted.

GreSetRectRegion

```
(BOOL) GreSetRectRegion (hdc, hrgn, parcRgn, cRect, pInstance, lFunction)
```

Sets the specified region to the region definition, given by an array of rectangles, unless the region is in use as a clipping region, in which instance an error is raised.

When no rectangles are specified (cRect is zero), GreSetRectRegion function produces an empty region.

Support

This function is supported by the graphics engine and can be hooked by the presentation driver.

Stack Frame

(HDC)	hdc	(device-context handle)
(HRGN)	hrgn	(region handle)
(PRECTL)	parcRgn	(far pointer to region definition, see below)
(LONG)	cRect	(count of rectangles in the region definition)
(ULONG)	pInstance	(pointer to instance data)
(ULONG)	lFunction	(High-order word = Flags, Low-order word = NGreSetRectRegion)

parcRgn the region is defined as an array of rectangles, each rectangle in the array is defined by a RECTL structure:

xLeft Minimum x coordinate of rectangle
yBottom Minimum y coordinate
xRight Maximum x coordinate of rectangle
yTop Maximum y coordinate.

The region is defined by the OR of all the rectangles.

For each rectangle, xRight must be equal to or greater than xLeft, and yTop must be equal to or greater than yBottom.

The bottom and left boundaries of each rectangle are part of the interior of the region. The top and right boundaries are not.

Return Codes

On completion, the handling routine must return BOOLEAN (fSuccess):

TRUE Successful.
FALSE Error.

When an error is detected, the handling routine must call WinSetErrorInfo to post the condition. Error codes for conditions that the handling routine is expected to check include:

PMERR_INSUFFICIENT_MEMORY
PMERR_INV_LENGTH_OR_COUNT
PMERR_INV_RECT
PMERR_INV_COORDINATE
PMERR_INV_HRGN
PMERR_HRGN_BUSY
PMERR_REGION_IS_CLIP_REGION.

```
(LONG) GreCombineRegion (hdc, hrgnDst, hrgnSrc1, hrgnSrc2, cmdMode, pInstance, lFunction)
```

Combines two regions to make a third. The destination region can be the same as one of the source regions. An error is raised when any one of the specified regions is currently selected as the clip region. All source and target regions must be of the same device class.

Support

This function is supported by the graphics engine and can be hooked by the presentation driver.

Stack Frame

(HDC)	hdc	(device-context handle)
(HRGN)	hrgnDst	(destination region handle)
(HRGN)	hrgnSrc1	(first region handle)
(HRGN)	hrgnSrc2	(second region handle)
(LONG)	cmdMode	(method of combination, see below)
(ULONG)	pInstance	(pointer to instance data)
(ULONG)	lFunction	(High-order word = Flags, Low-order word = NGreCombineRegion)

cmdMode Method of combination:

CRGN_OR Union of hrgnSrc1 and hrgnSrc2
CRGN_COPY hrgnSrc1 only (hrgnSrc2 ignored)
CRGN_XOR Symmetric difference of hrgnSrc1 and hrgnSrc2
CRGN_AND Intersection of hrgnSrc1 and hrgnSrc2
CRGN_DIFF hrgnSrc1 and NOT(hrgnSrc2).

Return Codes

On completion, the handling routine returns an integer (lComplexity) indicating the complexity of the new region:

RGN_ERROR Error
RGN_NULL Null region
RGN_RECT RECTangular region
RGN_COMPLEX COMPLEX region (more than 1 rectangle).

When an error is detected, the handling routine must call WinSetErrorInfo to post the condition. Error codes for conditions that the handling routine is expected to check include:

PMERR_INSUFFICIENT_MEMORY
PMERR_INV_LENGTH_OR_COUNT
PMERR_INV_RECT
PMERR_INV_COORDINATE
PMERR_INV_HRGN
PMERR_HRGN_BUSY
PMERR_REGION_IS_CLIP_REGION.

GreCombineRectRegion

```
(LONG) GreCombineRectRegion (hdc, hrgnDst, prcRect, hrgnSrc, cmdMode, pInstance, lFunction)
```

Combines a region with a rectangle to make a new region. The destination region can be the same as the source region.

An error is raised if either of the regions specified is currently selected as the clip region. The source and destination regions must be of the same device class.

Support

This function is supported by the graphics engine and can be hooked by the presentation driver.

Stack Frame

(HDC)	hdc	(device-context handle)
(HRGN)	hrgnDst	(destination region handle)
(PRECTL)	prcRect	(pointer to rectangle)
(HRGN)	hrgnSrc	(region handle)
(LONG)	cmdMode	(method of combination, see below)
(ULONG)	pInstance	(pointer to instance data)
(ULONG)	lFunction	(High-order word = Flags, Low-order word = NGreCombineRectRegion)

cmdMode Method of combination:

CRGN_OR	Union of hrgnSrc and prcRECT
CRGN_COPY	prcRECT only, hrgnSrc is ignored
CRGN_XOR	Symmetric difference of hrgnSrc and prcRECT
CRGN_AND	Intersection of hrgnSrc and prcRECT
CRGN_DIFF	hrgnSrc and NOT(prcRECT).

Return Codes

On completion, the handling routine returns an integer (lComplexity) indicating the complexity of the new region:

RGN_ERROR	Error
RGN_NULL	Null region
RGN_RECT	RECTangular region
RGN_COMPLEX	COMPLEX region (more than 1 rectangle).

When an error is detected, the handling routine must call WinSetErrorInfo to post the condition. Error codes for conditions that the handling routine is expected to check include:

PMERR_INV_REGION_MIX_MODE
PMERR_INSUFFICIENT_MEMORY
PMERR_INV_RECT
PMERR_INV_COORDINATE
PMERR_INV_HRGN
PMERR_HRGN_BUSY
PMERR_REGION_IS_CLIP_REGION.

```
(BOOL) GreCombineShortLineRegion (hdc, hrgn, ppsl, pInstance, lFunction)
```

Combines a polyshortline with a region. The polyshortline is ORed into the region. This function is used to build regions for path simulation.

An error is raised when the region specified is currently selected as the clip region.

Support

This function is supported by the graphics engine and can be hooked by the presentation driver.

Stack Frame

(HDC)	hdc	(device-context handle)
(HRGN)	hrgn	(region handle)
(PSHORTLINE)	ppsl	(long pointer to a polyshortline structure, see below)
(ULONG)	pInstance	(pointer to instance data)
(ULONG)	lFunction	(High-order word = Flags, Low-order word = NGreCombineShortLineRegion)

ppsl This is a long pointer to a polyshortline definition, which is a list of linked SHORTLINE structures:

slh SHORTLINEHEADER structure:

- **ulStyle** line style.
- **ptsStart** (x,y) position of start.
- **ptsStop** (x,y) position of end.
- **sxLeft** left edge of Box.
- **sxRight** right edge of Box.
- **pslhNext** pointer to next ShortLine.
- **pslhPrev** pointer to previous ShortLine.

This structure is a discrete representation of a curve that starts at point (x0,y0) and ends at point (x1,y1). For each of the (y1-y0+1) rows, there is exactly one x value contained in the x array − (the final point in the series is not drawn).

ax Steps.

It is guaranteed that the polyshortline does not intersect any part of the region.

Return Codes

This function returns BOOLEAN (fSuccess):

TRUE Successful.
FALSE Error.

When an error is detected, the handling routine must call WinSetErrorInfo to post the condition. Error codes for conditions that the handling routine is expected to check include:

PMERR_INSUFFICIENT_MEMORY
PMERR_INV_HRGN
PMERR_HRGN_BUSY
PMERR_REGION_IS_CLIP_REGION.

GreEqualRegion

```
(LONG) GreEqualRegion (hdc, hrgnSrc1, hrgnSrc2, pInstance, lFunction)
```

Checks whether two regions, owned by the device identified by hdc, are identical.

An error is raised when either region is currently selected as the clip region.

Support

This function is supported by the graphics engine and can be hooked by the presentation driver.

Stack Frame

(HDC)	hdc	(device-context handle)
(HRGN)	hrgnSrc1	(first region handle)
(HRGN)	hrgnSrc2	(second region handle)
(ULONG)	pInstance	(pointer to instance data)
(ULONG)	lFunction	(High-order word = Flags, Low-order word = NGreEqualRegion)

Return Codes

On completion, the handling routine returns an integer (lEquality) indicating whether the regions are equal:

EQRGN_ERROR Error
EQRGN_NOTEQUAL Not equal
EQRGN_EQUAL Equal.

When an error is detected, the handling routine must call WinSetErrorInfo to post the condition. Error codes for conditions that the handling routine is expected to check include:

PMERR_INV_HRGN
PMERR_HRGN_BUSY
PMERR_REGION_IS_CLIP_REGION.

```
(LONG) GrePaintRegion (hdc, hrgn, pInstance, lFunction)
```

Paints the specified region using the current area foreground and background colors. Mixing is controlled by the area foreground mix only.

This function raises an error when the region is currently selected as a clip region.

Note: If the preprocessor in the graphics engine receives a GrePaintRegion call for a NULL region, the preprocessor returns GPI_OK and does not forward the call through the dispatch table to the handling routine. Presentation drivers for devices that do not support bitblt operations, that is for vector devices such as plotters, should not hook this function; such drivers would only hook the function to return an error code.

Support

This function is supported by the graphics engine and can be hooked by the presentation driver.

Stack Frame

(HDC)	hdc	(device-context handle)
(HRGN)	hrgn	(region handle)
(ULONG)	pInstance	(pointer to instance data)
(ULONG)	lFunction	(High-order word = Flags, Low-order word = NGrePaintRegion)

Return Codes

On completion, the handling routine must return an integer (cHits) indicating, where appropriate, whether correlation hits were detected:

GPI_OK Successful.
GPI_HITS Successful with correlate hit (returned by display drivers when the correlate flag is on and a hit is detected).
GPI_ERROR Error.

When an error is detected, the handling routine must call WinSetErrorInfo to post the condition. Error codes for conditions that the handling routine is expected to check include:

PMERR_INV_IN_AREA
PMERR_INV_IN_PATH
PMERR_HDC_BUSY
PMERR_INV_HDC
PMERR_INV_HRGN
PMERR_HRGN_BUSY
PMERR_REGION_IS_CLIP_REGION.

Clip Functions

- GreGetClipBox
- GreGetClipRects
- GreOffsetClipRegion
- GrePtVisible
- GreRectVisible
- GreQueryClipRegion
- GreSelectClipRegion
- GreIntersectClipRectangle
- GreExcludeClipRectangle
- GreSetXformRect
- GreSaveRegion
- GreRestoreRegion
- SelectPathRegion
- GreRegionSelectBitmap
- GreCopyClipRegion
- GreSetupDC.

Clip regions are defined as rectangles, in world coordinates. The boundaries of a clip-region rectangle are inclusive of the rectangle they define.

```
(LONG) GreGetClipBox (hdc, prcRect, pInstance, lFunction)
```

Determines the dimensions of the tightest rectangle around the DC region in world coordinates.

The DC region is defined as the intersection of the visible region, clip region, viewing limits, graphics field, and clip path. The bounding rectangle is inclusive, that is, points on its boundary are deemed to be inside it.

prcRect is a null rectangle when the value of xRight is less than xLeft and yTop is less than yBottom.

Support

This function is supported by the graphics engine and can be hooked by the presentation driver.

Stack Frame

(HDC)	hdc	(device context handle)
(PRECTL)	prcRect	(pointer to rectangle in world coordinates, see below)
(ULONG)	pInstance	(pointer to instance data)
(ULONG)	lFunction	(High-order word = Flags, Low-order word = NGreGetClipBox)

prcRect RECTL structure:

xLeft Minimum x coordinate of rectangle
yBottom Minimum y coordinate
xRight Maximum x coordinate of rectangle
yTop Maximum y coordinate.

Return Codes

The handling routine should return an integer (lComplexity) indicating the complexity of the resultant clipping region, which is defined as the intersection of all clipping; that is, the clip path, viewing limits, graphics field, clip region, and visible region.

RGN_ERROR Error
RGN_NULL Null region
RGN_RECT RECTangular region
RGN_COMPLEX COMPLEX region (more than 1 rectangle).

When an error is detected, the handling routine must call WinSetErrorInfo to post the condition. Error codes for conditions that the handling routine is expected to check include:

PMERR_HDC_BUSY
PMERR_DEV_FUNC_NOT_INSTALLED
PMERR_COORDINATE_OVERFLOW
PMERR_INV_LENGTH_OR_COUNT
PMERR_INV_RECT
PMERR_INV_HDC
PMERR_INV_COORD_SPACE.

GreGetClipRects

```
(BOOL) GreGetClipRects (hdc, prcBound, pControl, parcRect, pInstance, lFunction)
```

Fills the buffer indicated by parcRect with a list of (x,y) coordinate pairs specifying the rectangles which define the DC region and intersect an optional bounding rectangle (prcRect).

Clipping, when done by the graphics engine carries a heavy processing overhead. For simple clipping, the presentation driver can gain a significant performance advantage by calling this function to enumerate the clip region, and to clip the line to each rectangle in turn.

The DC region is the intersection of the visible region, clip region, viewing limits, graphics field, and clip path.

Support

This function is supported by the graphics engine and can be hooked by the presentation driver.

Stack Frame

(HDC)	hdc	(device context handle)
(PRECTL)	prcBound	(pointer to bounding rectangle, in device coordinates, see below)
(PRGNRECT)	pControl	(pointer to control structure, see below)
(PRECTL)	parcRect	(pointer to array of rectangle, RECTL, structures)
(ULONG)	pInstance	(pointer to instance data)
(ULONG)	lFunction	(High-order word = Flags, Low-order word = NGreGetClipRects)

prcBound Pointer to a RECTL structure:

xLeft Minimum x coordinate of rectangle
yBottom Minimum y coordinate
xRight Maximum x coordinate of rectangle
yTop Maximum y coordinate.

Only rectangles intersecting this bounding rectangle are returned. When this pointer is NULL, all rectangles in the region are enumerated. When this pointer is not NULL, each rectangle returned is the intersection of the bounding rectangle with a rectangle in the region.

pControl Pointer to a control structure containing additional parameters:

lStart The rectangle number to start enumerating (one means start at the beginning).

By updating this value, the function can be called repeatedly to allow for more rectangles than can be stored in the receiving buffer.

crc The number of rectangles that can fit into the buffer (at least 1).

crcReturned Number of rectangles that were written into the buffer (if less than lStart, there are no more rectangles to enumerate).

usDirection The direction in which the rectangles are listed:
RECTDIR_LFRT_TOPBOT Left to right, top to bottom
RECTDIR_RTLF_TOPBOT Right to left, top to bottom
RECTDIR_LFRT_BOTTOP Left to right, bottom to top
RECTDIR_RTLF_BOTTOP Right to left, bottom to top.

Return Codes

The handling routine should return an BOOLEAN (fSuccess):

TRUE Successful completion.
FALSE Error.

When an error is detected, the handling routine must call WinSetErrorInfo to post the condition. Error codes for conditions that the handling routine is expected to check include:

PMERR_HDC_BUSY
PMERR_DEV_FUNC_NOT_INSTALLED
PMERR_INV_RECT
PMERR_INV_COORDINATE
PMERR_INV_HDC
PMERR_INV_REGION_CONTROL.

GreOffsetClipRegion

```
(LONG) GreOffsetClipRegion (hdc, pdpt, pInstance, lFunction)
```

Moves the clipping region by the specified amount.

The value returned is the complexity of the resultant DC region. That is, the intersection of all clipping, such as, clip path, viewing limits, graphics field, clip region, and visible region.

Support

This function is supported by the graphics engine and can be hooked by the presentation driver.

Stack Frame

(HDC)	hdc	(device context handle)
(PPOINTL)	pdpt	(pointer to offset by which clipping region is to be moved, see below)
(ULONG)	pInstance	(pointer to instance data)
(ULONG)	lFunction	(High-order word = Flags, Low-order word = NGreOffsetClipRegion)

pdpt Offsets by which the clip region is to be moved, in world coordinates:

dx x offset
dy y offset.

Return Codes

The handling routine should return an integer (lComplexity) indicating the complexity of the region:

RGN_ERROR Error
RGN_NULL Null region
RGN_RECT RECTangular region
RGN_COMPLEX COMPLEX region (more than 1 rectangle).

When an error is detected, the handling routine must call WinSetErrorInfo to post the condition. Error codes for conditions that the handling routine is expected to check include:

PMERR_INV_IN_AREA
PMERR_INV_IN_PATH
PMERR_HDC_BUSY
PMERR_DEV_FUNC_NOT_INSTALLED
PMERR_COORDINATE_OVERFLOW
PMERR_INSUFFICIENT_MEMORY
PMERR_INV_LENGTH_OR_COUNT
PMERR_INV_RECT
PMERR_INV_COORDINATE
PMERR_INV_HDC
PMERR_INV_HRGN
PMERR_INV_REGION_CONTROL
PMERR_INV_COORD_OFFSET
PMERR_INV_COORD_SPACE.

```
(LONG) GrePtVisible (hdc, pptPoint, pInstance, lFunction)
```

Checks whether a point is visible within the DC region of the specified device context. The DC region is defined as the intersection of application clipping and the clipping resulting from windowing.

Support

This function is supported by the graphics engine and can be hooked by the presentation driver.

Stack Frame

(HDC)	hdc	(device context handle)
(PPOINTL)	pptPoint	(pointer to (x,y) point in world coordinates)
(ULONG)	pInstance	(pointer to instance data)
(ULONG)	lFunction	(High-order word = Flags, Low-order word = NGrePtVisible)

Return Codes

The handling routine should return an integer (lVisible) indicating the visibility of the point:

PVIS_ERROR Error
PVIS_INVISIBLE Not visible
PVIS_VISIBLE Visible.

When an error is detected, the handling routine must call WinSetErrorInfo to post the condition. Error codes for conditions that the handling routine is expected to check include:

PMERR_HDC_BUSY
PMERR_DEV_FUNC_NOT_INSTALLED
PMERR_COORDINATE_OVERFLOW
PMERR_INV_LENGTH_OR_COUNT
PMERR_INV_RECT
PMERR_INV_COORDINATE
PMERR_INV_HDC
PMERR_INV_COORD_SPACE.

GreRectVisible

```
(LONG) GreRectVisible (hdc, prcRect, pInstance, lFunction)
```

Checks whether any part of the given rectangle is visible within the DC region. The DC region is the intersection of application clipping and clipping resulting from windowing.

Support

This function is supported by the graphics engine and can be hooked by the presentation driver.

Stack Frame

(HDC)	hdc	(device context handle)
(PRECTL)	prcRect	(pointer to rectangle, in world coordinates, see below)
(ULONG)	pInstance	(pointer to instance data)
(ULONG)	lFunction	(High-order word = Flags, Low-order word = NGreRectVisible)

prcRect RECTL structure:

xLeft Minimum x coordinate of rectangle
yBottom Minimum y coordinate
xRight Maximum x coordinate of rectangle
yTop Maximum y coordinate.

Return Codes

The handling routine should return an integer (lVisible) indicating the visibility of the rectangle.

RVIS_ERROR Error
RVIS_INVISIBLE Not visible
RVIS_PARTIAL Partially visible
RVIS_VISIBLE All visible.

When an error is detected, the handling routine must call WinSetErrorInfo to post the condition. Error codes for conditions that the handling routine is expected to check include:

PMERR_HDC_BUSY
PMERR_DEV_FUNC_NOT_INSTALLED
PMERR_COORDINATE_OVERFLOW
PMERR_INV_LENGTH_OR_COUNT
PMERR_INV_RECT
PMERR_INV_COORDINATE
PMERR_INV_HDC
PMERR_INV_COORD_SPACE.

```
(HRGN) GreQueryClipRegion (hdc, pInstance, lFunction)
```

Returns the handle of the currently selected clip region (or NULL if none exists).

Support

This function is supported by the graphics engine and can be hooked by the presentation driver.

Stack Frame

(HDC)	hdc	(device context handle)
(ULONG)	pInstance	(pointer to instance data)
(ULONG)	lFunction	(High-order word = Flags, Low-order word = NGreQueryClipRegion)

Return Codes

The handling routine must return the handle of the clip region:

hrgn	Region handle
NULL	Null handle (no region selected)
HRGN_ERROR	Error.

When an error is detected, the handling routine must call WinSetErrorInfo to post the condition. Error codes for conditions that the handling routine is expected to check include:

PMERR_HDC_BUSY
PMERR_INV_HDC.

GreSelectClipRegion

```
(LONG) GreSelectClipRegion (hdc, hrgn, phrgnOld, pInstance, lFunction)
```

Specifies the region to be used for clipping. A region can only be selected by one DC at a time. Once a region is selected, operations that reference its handle are invalid.

The coordinates of the region are in device coordinates within the DC. Clipping is inclusive at the left and bottom boundaries and exclusive at the right and top boundaries.

Functions that modify the clipping region also modify the region when its handle is returned by a subsequent GreSelectClipRegion.

GreSelectClipRegion passes the handle of the previously selected clip region back to the calling routine at the location addressed by phrgnOld. If this handle is not null, the calling routine is responsible for destroying the previous clip region; the clip region should be destroyed regardless of whether it was created explicitly or in a call to GreIntersectClipRectangle or GreExcludeClipRectangle. (If the default clip region is being used when GreSelectClipRegion is called, a null region handle is returned.)

Support

This function is supported by the graphics engine and can be hooked by the presentation driver.

Stack Frame

(HDC)	hdc	(device context handle)
(HRGN)	hrgn	(region handle, see below)
(PHRGN)	phrgnOld	(pointer to old region handle)
(ULONG)	pInstance	(pointer to instance data)
(ULONG)	lFunction	(High-order word = Flags, Low-order word = NGreSelectClipRegion)

hrgn When NULL, the clip region is set to no clipping, the initial state.
phrgnOld Previously selected region, if this is a NULL handle, there was no clip region.

Return Codes

The handling routine must return an integer (lComplexity) indicating the complexity of the DC region:

RGN_ERROR Error
RGN_NULL Null region
RGN_RECT RECTangular region
RGN_COMPLEX COMPLEX region (more than 1 rectangle).

When an error is detected, the handling routine must call WinSetErrorInfo to post the condition. Error codes for conditions that the handling routine is expected to check include:

PMERR_INV_IN_AREA
PMERR_INV_IN_PATH
PMERR_HDC_BUSY
PMERR_DEV_FUNC_NOT_INSTALLED
PMERR_INSUFFICIENT_MEMORY
PMERR_INV_RECT
PMERR_INV_COORDINATE
PMERR_INV_HDC
PMERR_INV_HRGN
PMERR_HRGN_BUSY
PMERR_REGION_IS_CLIP_REGION
PMERR_INV_REGION_CONTROL.

```
(LONG) GreIntersectClipRectangle (hdc, prcRect, pInstance, lFunction)
```

Sets the new clipping region to the intersection of the current clip region and the specified rectangle.

When no clip region exists, this function must create one. The application must then free the handle when it subsequently selects another clip region. The return value of this function is the complexity of resultant DC region. This is defined as the intersection of all clipping.

Support

This function is supported by the graphics engine and can be hooked by the presentation driver.

Stack Frame

(HDC)	hdc	(device context handle)
(PRECTL)	prcRect	(pointer to rectangle in world coordinates, see below)
(ULONG)	pInstance	(pointer to instance data)
(ULONG)	lFunction	(High-order word = Flags, Low-order word = NGreIntersectClipRectangle)

prcRect RECTL structure, defined in world coordinates:

xLeft Minimum x coordinate of rectangle
yBottom Minimum y coordinate
xRight Maximum x coordinate of rectangle
yTop Maximum y coordinate.

All the boundaries of the rectangle are inclusive (part of the interior) and are not clipped.

Return Codes

The handling routine must return an integer (lComplexity) indicating the complexity of the DC region.

RGN_ERROR Error
RGN_NULL Null region
RGN_RECT RECTangular region
RGN_COMPLEX COMPLEX region (more than 1 rectangle).

When an error is detected, the handling routine must call WinSetErrorInfo to post the condition. Error codes for conditions that the handling routine is expected to check include:

PMERR_INV_IN_AREA
PMERR_INV_IN_PATH
PMERR_HDC_BUSY
PMERR_DEV_FUNC_NOT_INSTALLED
PMERR_COORDINATE_OVERFLOW
PMERR_INSUFFICIENT_MEMORY
PMERR_INV_LENGTH_OR_COUNT
PMERR_INV_RECT
PMERR_INV_COORDINATE
PMERR_INV_HDC
PMERR_INV_HRGN
PMERR_INV_REGION_CONTROL
PMERR_INV_COORD_SPACE.

GreExcludeClipRectangle

```
(LONG) GreExcludeClipRectangle (hdc, prcRect, pInstance, lFunction)
```

Excludes the specified rectangle from the clipping region. The points on the boundary and inside the rectangle are clipped and are removed from the clip region. This function creates a clip region, when none exists. The application is responsible for destroying this clip region when it has finished, otherwise it is not destroyed until the DC is closed.

Support

This function is supported by the graphics engine and can be hooked by the presentation driver.

Stack Frame

(HDC)	hdc	(device context handle)
(PRECTL)	prcRect	(pointer to rectangle in world coordinates, see below)
(ULONG)	pInstance	(pointer to instance data)
(ULONG)	lFunction	(High-order word = Flags, Low-order word = NGreExcludeClipRectangle)

prcRect RECTL structure:

xLeft Minimum x coordinate of rectangle
yBottom Minimum y coordinate
xRight Maximum x coordinate of rectangle
yTop Maximum y coordinate.

All the boundaries of this rectangle are considered to be part of the interior and are clipped accordingly.

Return Codes

The handling routine must return an integer (lComplexity) indicating the complexity of the DC region:

RGN_ERROR Error
RGN_NULL Null region
RGN_RECT RECTangular region
RGN_COMPLEX COMPLEX region (more than 1 rectangle).

When an error is detected, the handling routine must call WinSetErrorInfo to post the condition. Error codes for conditions that the handling routine is expected to check include: function include:

PMERR_INV_IN_AREA
PMERR_INV_IN_PATH
PMERR_HDC_BUSY
PMERR_DEV_FUNC_NOT_INSTALLED
PMERR_COORDINATE_OVERFLOW
PMERR_INSUFFICIENT_MEMORY
PMERR_INV_LENGTH_OR_COUNT
PMERR_INV_RECT
PMERR_INV_COORDINATE
PMERR_INV_HDC
PMERR_INV_HRGN
PMERR_INV_REGION_CONTROL
PMERR_INV_COORD_SPACE.

```
(LONG) GreSetXFormRect (hdc, prcRect, pInstance, lFunction)
```

Intersects a rectangle in device coordinates with the DC region. The rectangle is inclusive at the bottom and left boundaries and exclusive at the top and right boundaries.

Support

This function is supported by the graphics engine and can be hooked by the presentation driver.

Stack Frame

(HDC)	hdc	(device context handle)
(PRECTL)	prcRect	(pointer to a rectangle, in device coordinates)
(ULONG)	pInstance	(pointer to instance data)
(ULONG)	lFunction	(High-order word = Flags, Low-order word = NGreSetXFormRect)

prcRect RECTL structure:

xLeft Minimum x coordinate of rectangle
yBottom Minimum y coordinate
xRight Maximum x coordinate of rectangle
yTop Maximum y coordinate.

Return Codes

The handling routine must return an integer (lComplexity) indicating the complexity of the DC region:

RGN_ERROR Error
RGN_NULL Null region
RGN_RECT RECTangular region
RGN_COMPLEX COMPLEX region (more than 1 rectangle).

When an error is detected, the handling routine must call WinSetErrorInfo to post the condition. Error codes for conditions that the handling routine is expected to check include:

PMERR_HDC_BUSY
PMERR_DEV_FUNC_NOT_INSTALLED
PMERR_INSUFFICIENT_MEMORY
PMERR_INV_RECT
PMERR_INV_COORDINATE
PMERR_INV_HDC
PMERR_INV_HRGN
PMERR_INV_REGION_CONTROL.

GreSaveRegion

```
(BOOL) GreSaveRegion (hdc, cSave, pInstance, lFunction)
```

This function is called during SaveDC and OpenDC to allow simulations to save their local data structures.

When a new DC is created, this function is called with a save level of 1 to initialize its local data.

A save level with a value of one more than the current, generates intervening levels. A negative save value is invalid.

When initializing a new DC, a valid visible region handle must be created.

Support

This function is supported by the graphics engine, it is hookable by the presentation driver and is required in case an installed simulation is in use.

Stack Frame

(HDC)	hdc	(device context handle)
(LONG)	cSave	(DC save level, this must not be a negative value)
(ULONG)	pInstance	(pointer to instance data)
(ULONG)	lFunction	(High-order word = Flags, Low-order word = NGreSaveRegion)

Return Codes

On completion, the handling routine must return BOOL (fSuccess):

TRUE Successful.
FALSE Error.

When an error is detected, the handling routine must call WinSetErrorInfo to post the condition. Error codes for conditions that the handling routine is expected to check include:

PMERR_HDC_BUSY
PMERR_INSUFFICIENT_MEMORY
PMERR_INV_RECT
PMERR_INV_COORDINATE
PMERR_INV_HDC
PMERR_INV_HRGN.

```
(BOOL) GreRestoreRegion (hdc, cSave, pInstance, lFunction)
```

This function is called during RestoreDC and CloseDC to allow simulations to restore their local data structures.

When a DC is closed, this function is called with a count of zero to free its local data.

The clip region currently selected into the DC is deleted by this function.

Support

This function is supported by the graphics engine and can be hooked by the presentation driver.

Stack Frame

(HDC)	hdc	(device context handle)
(LONG)	cSave	(DC save level, see below)
(ULONG)	pInstance	(pointer to instance data)
(ULONG)	lFunction	(High-order word = Flags, Low-order word = NGreRestoreRegion)

cSave Indicates the saved DC state that the handling routine should use to restore the region.

When this is passed as −1, the handling routine resets the region to its initial state (−1 is passed to this routine from GreResetDC and is the only valid negative value). A value of zero also indicates that the region should be restored to its initial state; other positive values identify which saved level should be restored (see "Enable subfunction 08H − RestoreDCState" on page 11-16).

Return Codes

On completion this function must return BOOL (fSuccess):

TRUE Successful
FALSE Error.

When an error is detected, the handling routine must call WinSetErrorInfo to post the condition. Error codes for conditions that the handling routine is expected to check include:

PMERR_HDC_BUSY
PMERR_DEV_FUNC_NOT_INSTALLED
PMERR_INSUFFICIENT_MEMORY
PMERR_INV_RECT
PMERR_INV_COORDINATE
PMERR_INV_HDC
PMERR_INV_HRGN
PMERR_INV_REGION_CONTROL.

GreSelectPathRegion

```
(BOOL) GreSelectPathRegion  (hdc, hrgn, pInstance, lFunction)
```

Merges a region, representing a path, into the DC region.

A path region behaves exactly like any other region. It has no special properties relating to paths.

Support

This function is supported by the graphics engine.

Stack Frame

(HDC)	hdc	(device context handle)
(HRGN)	hrgn	(region handle)
(ULONG)	pInstance	(instance data from the device context)
(ULONG)	lFunction	(High-order word = Flags, Low-order word = NGreSelectPathRegion)

Return Codes

This function returns BOOLEAN (fSuccess):

TRUE Successful.
FALSE Error.

When an error is detected, the handling routine must call WinSetErrorInfo to post the condition. Error codes for conditions that the handling routine is expected to check include:

PMERR_HDC_BUSY
PMERR_DEV_FUNC_NOT_INSTALLED
PMERR_INSUFFICIENT_MEMORY
PMERR_INV_RECT
PMERR_INV_COORDINATE
PMERR_INV_HDC
PMERR_INV_HRGN
PMERR_HRGN_BUSY
PMERR_REGION_IS_CLIP_REGION
PMERR_INV_REGION_CONTROL.

```
(BOOL) GreRegionSelectBitmap (hdc, hbm, pInstance, lFunction)
```

This function is called when a new bit-map handle is selected into a memory DC. It removes the old visible region and informs the presentation driver that the DC region must be recalculated.

Note: In a nonwindowed DC, the visible region represents the device boundary.

Support

This function is supported by the engine and can be hooked by the presentation driver.

Stack Frame

(HDC)	hdc	(device context handle)
(HBITMAP)	hbm	(handle of bit map to be selected)
(ULONG)	pInstance	(pointer to instance data)
(ULONG)	lFunction	(High-order word = Flags, Low-order word = NGreRegionSelectBitmap)

Return Codes

This function returns BOOLEAN (fSuccess):

TRUE Successful.
FALSE Error.

When an error is detected, the handling routine must call WinSetErrorInfo to post the condition. Error codes for conditions that the handling routine is expected to check include:

PMERR_HDC_BUSY
PMERR_DEV_FUNC_NOT_INSTALLED
PMERR_INSUFFICIENT_MEMORY
PMERR_INV_RECT
PMERR_INV_COORDINATE
PMERR_INV_HDC
PMERR_INV_HRGN
PMERR_INV_REGION_CONTROL.

GreCopyClipRegion

```
(LONG) GreCopyClipRegion (hdc, hrgn, prcBounds, flOptions, pInstance, lFunction)
```

Copies the visible region, the clip region, or the DC region and returns the complexity and bounds of the resulting region.

Support

This function is supported by the engine and can be hooked by the presentation driver.

Stack Frame

(HDC)	hdc	(device context handle)
(HRGN)	hrgn	(visible-region handle)
(PRECTL)	prcBounds	(bounding rectangle)
(ULONG)	flOptions	(option flags)
(ULONG)	pInstance	(pointer to instance data)
(ULONG)	lFunction	(High-order word = Flags, Low-order word = NGreCopyClipRegion)

prcBounds Pointer to the bounding rectangle of the returned region. The bounding rectangle is returned in the same coordinate system as region, defined by flOptions. This rectangle is inclusive at the bottom and left boundaries and exclusive at the top and right boundaries. When this is specified as NULL, the bounding rectangle is not returned.

flOptions These flags determine the type of region to be returned in hrgn:

COPYCRGN_ALLINTERSECT The function must return the intersection of all clipping. This describes the DC region and is expressed in screen coordinates.

COPYCRGN_VISRGN The function must return a copy of the visible region only. This is returned in screen coordinates.

COPYCRGN_CLIPRGN The function must return a copy of the clip region only. The clip region is expressed in device coordinates.

Return Codes

On completion, the handling routine returns an integer (lComplexity), indicating the complexity of the region:

RGN_ERROR Error
RGN_NULL Null region
RGN_RECT RECTangular region
RGN_COMPLEX COMPLEX region (more than 1 rectangle).

When an error is detected, the handling routine must call WinSetErrorInfo to post the condition. Error codes for conditions that the handling routine is expected to check include:

PMERR_HDC_BUSY
PMERR_DEV_FUNC_NOT_INSTALLED
PMERR_INSUFFICIENT_MEMORY
PMERR_INV_RECT
PMERR_INV_COORDINATE
PMERR_INV_HDC
PMERR_INV_HRGN
PMERR_HRGN_BUSY
PMERR_REGION_IS_CLIP_REGION.

```
(BOOL) GreSetupDC (hdc, hrgnVis, xOrg, yOrg, prcBounds, flOptions, pInstance, lFunction)
```

Initializes the device context to the region determined by flOptions.

Support

This function is supported by the graphics engine and can be hooked by the presentation driver.

Stack Frame

(HDC)	hdc	(device context handle)
(HRGN)	hrgnVis	(visible-region handle)
(LONG)	xOrg	(x coordinate of DC origin specified in screen coordinates)
(LONG)	yOrg	(y coordinate of DC origin specified in screen coordinates)
(PRECTL)	prcBounds	(bounding rectangle, in device coordinates)
(ULONG)	flOptions	(option flags, see below)
(ULONG)	pInstance	(pointer to instance data)
(ULONG)	lFunction	(High-order word = Flags, Low-order word = NGreSetupDC)

flOptions These flags determine the region returned in prcBounds:

SETUPDC_VISRGN	Replace the contents of the visible region of hdc with the contents of hrgnVis.
SETUPDC_ORIGIN	Set the DC origin to (xOrg, yOrg).
SETUPDC_ACCUMBOUNDSON	Turn on bounds accumulation. This only affects the COM_ALT_BOUND flag. If the COM_ALT_BOUND (see 10-5) is not set, the bounds rectangle is reset to an empty rectangle. If the COM_ALT_BOUND is already set, the bounds rectangle is not changed.
SETUPDC_ACCUMBOUNDSOFF	Turn off bounds accumulation.
SETUPDC_REALCLIP	Recalculate the true device clipping region. This bit is normally set, but can be zero when immediate recalculation is not required.
SETUPDC_SETOWNER	When this bit is set, the DC must belong to the current process.
SETUPDC_CLEANDC	When this bit is set, the simulation marks the visible regions as valid and calls GreNotifyClipChange in the presentation driver.

Return Codes

On completion this function must return BOOL (fSuccess):

TRUE	Successful
FALSE	Error.

When an error is detected, the handling routine must call WinSetErrorInfo to post the condition. Error codes for conditions that the handling routine is expected to check include:

PMERR_HDC_BUSY
PMERR_DEV_FUNC_NOT_INSTALLED
PMERR_COORDINATE_OVERFLOW
PMERR_INSUFFICIENT_MEMORY
PMERR_INV_LENGTH_OR_COUNT
PMERR_INV_RECT
PMERR_INV_COORDINATE
PMERR_INV_HDC
PMERR_INV_HRGN
PMERR_HRGN_BUSY
PMERR_REGION_IS_CLIP_REGION
PMERR_INV_REGION_CONTROL
PMERR_INV_COORD_SPACE.

Transform Functions

- GreGetPageUnits
- GreSetPageUnits
- GreGetModelXform
- GreSetModelXform
- GreGetWindowViewportXform
- GreSetWindowViewportXform
- GreGetGlobalViewingXform
- GreSetGlobalViewingXform
- GreSaveXformData
- GreRestoreXformData
- GreGetPageViewport
- GreSetPageViewport
- GreGetGraphicsField
- GreSetGraphicsField
- GreGetViewingLimits
- GreSetViewingLimits
- GreGetDefaultViewingLimits
- GreSetDefaultViewingLimits
- GreQueryViewportSize
- GreConvert
- GreSaveXform
- GreRestoreXform
- GreMultiplyXforms
- GreConvertWithMatrix.

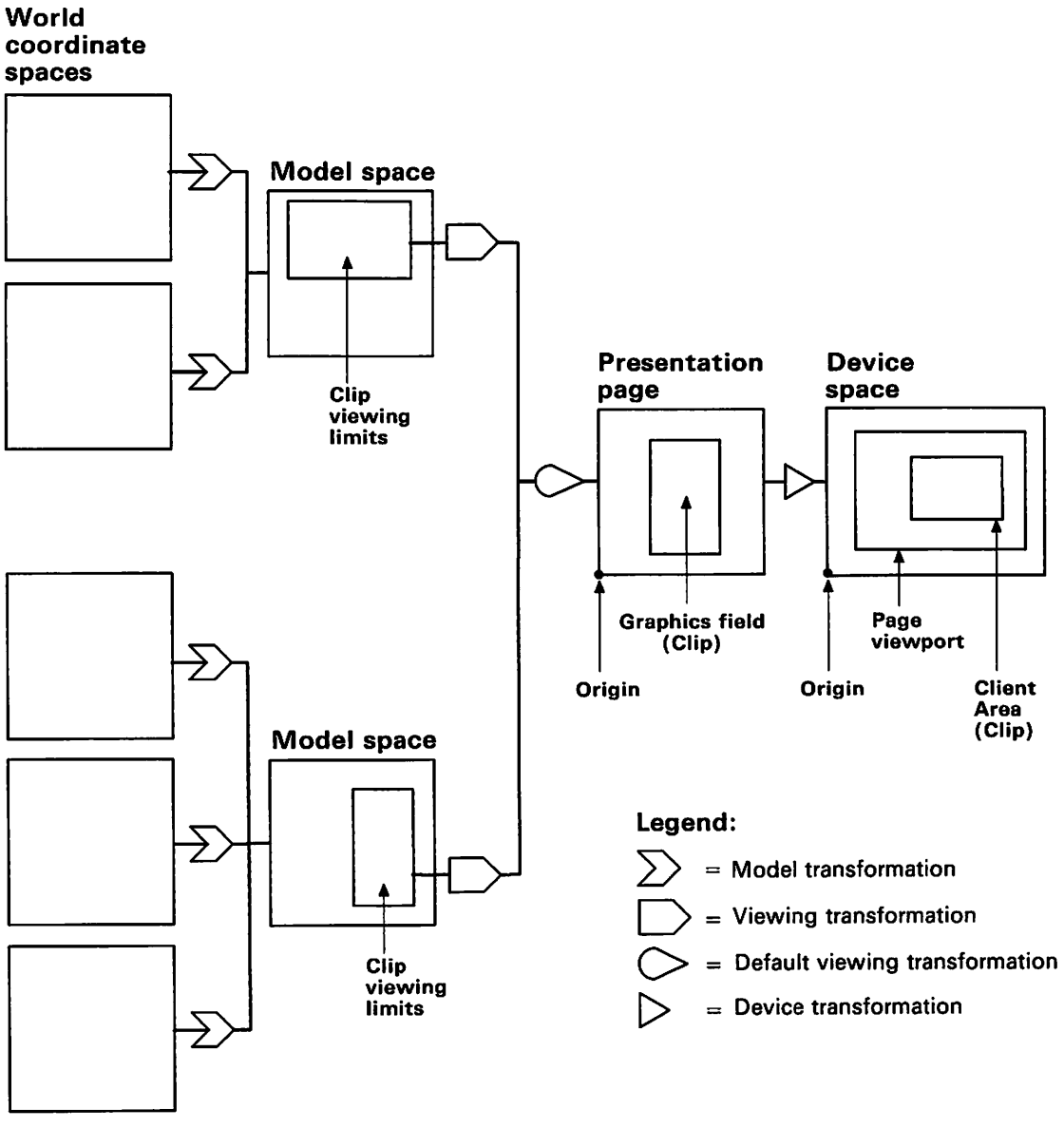


Figure 14-1. Transform and Clipping Pipeline

Matrix Element Format

The matrix elements for the model and viewing transforms are held in XFORM structures:

fxM11
fxM12
fxM21
fxM22
lM41
lM42.

M11, M12, M21, and M22 are fixed-point numbers represented as signed 4-byte integers with a notional binary point between bits 16 and 15:

```
+2.5 is represented by 00028000H
-2.5 is represented by FFFD8000H
-0.5 is represented by FFFF8000H
```

M41 and M42 are signed 4-byte numbers.

Device Transform Definition by Presentation Page Viewport

For a transform, defined by viewport and window rectangles whose bottom-left and top-right coordinates are represented by (X1, Y1), (X2, Y2), (X3, Y3), (X4, Y4) respectively, the matrix elements are determined as follows:

The point (X3-1/2, Y3-1/2) transforms to (X1-1/2, Y1-1/2) and (X4+1/2, Y4+1/2) transforms to (X2+1/2, Y2+1/2). Thus:

```
M12 = 0
M21 = 0

If  X4 >= X3 then
M11 = (X2 - X1 + 1) / (X4 - X3 + 1)
M41 = (X1 * X4 - X3 * X2 + 1/2 * (X2 - X4 + X1 - X3)) / (X4 - X3 + 1)

If  X4 < X3 then
M11 = (X2 - X1 + 1) / (X4 - X3 - 1)
M41 = (X1 * X4 - X3 * X2 - 1/2 * (X2 + X4 + X1 + X3)) / (X4 - X3 - 1)

If  Y4 >= Y3 then
M22 = (Y2 - Y1 + 1) / (Y4 - Y3 + 1)
M42 = ((Y1 * Y4 - Y3 * Y2 + 1/2 * (Y2 - Y4 + Y1 - Y3)) / (Y4 - Y3 + 1)

If  Y4 < Y3 then
M22 = (Y2 - Y1 + 1) / (Y4 - Y3 - 1)
M42 = (Y1 * Y4 - Y3 * Y2 - 1/2 * (Y2 + Y4 + Y1 + Y3)) / (Y4 - Y3 - 1)
```

Note: X4 is always greater than X3 and Y4 is always greater than Y3.

In the case of device transforms, (X3,Y3) is always (0,0). Y4 is always greater than Y3, and the device space coordinates (X2,Y2) are exclusive. This simplifies the formulae to:

```
M12=0
M21=0
M11 = (X2 -X1) / (X4 +1)
M41 = (X1*X4 + 1/2*(X2 -1 - X4+ X1)) / (X4 +1) = X1 + 1/2(M11 -1)
M22 = (Y2 -Y1) / (Y4 +1)
M42 = (Y1*Y4 + 1/2*(Y2 -1 - Y4+ Y1)) / (Y4 +1) = Y1 + 1/2(M22 -1)
```

Bounds, Correlation, and Clipping

All presentation drivers must support bounds computation for both GPI bounds (COM_BOUND) and user bounds (COM_ALT_BOUND). Bounds are calculated, on unclipped primitives, for all operations that draw to the device, including AVIO functions.

GPI bounds are passed in model-space coordinates, user bounds are calculated in device-coordinate space. To prevent inaccuracies occurring when the transform changes, the typical driver maintains bounds in both coordinate sets in its instance data structure. It then accumulates the transforms as they occur.

Correlation is performed, in page-coordinate space, on the output of primitives that have been clipped to the viewing limits and graphics field only. Correlation is also done on all operations that draw to the device, except the AVIO function.

Printer device drivers are not required to perform correlation.

GreGetPageUnits

```
(ULONG) GreGetPageUnits (hdc, pExtent, pInstance, lFunction)
```

Returns the page units for the specified display context. On completion, the structure addressed by pExtent contains the dimensions of the page, expressed in the units indicated by the return value of this function.

Support

This function is supported by the graphics engine and can be hooked by the presentation driver.

Stack Frame

(HDC)	hdc	(device-context handle)
(PSIZEL)	pExtent	(points to a SIZEL structure where the page dimensions are returned)
(ULONG)	pInstance	(pointer to instance data)
(ULONG)	lFunction	(High-order word = Flags, Low-order word = NGreGetPageUnits)

pExtent On completion, the SIZEL structure contains the actual dimensions of the page:

lWidth Page width.
lHeight Page height.

Return Codes

On completion, the handling routine returns the page units or GPI_ERROR if an error occurred. The defined page units are described under "GreSetPageUnits" on page 14-83.

When an error is detected, the handling routine must call WinSetErrorInfo to post the condition. Error codes for conditions that the handling routine is expected to check include:

PMERR_HDC_BUSY
PMERR_INV_HDC.

```
(BOOL) GreSetPageUnits (hdc, ulUnits, lWidth, lHeight, pInstance, lFunction)
```

Sets the page units controlling the device transform.

Support

This function is supported by the graphics engine and can be hooked by the presentation driver.

Stack Frame

(HDC)	hdc	(device-context handle)
(ULONG)	ulUnits	(page units, see below)
(LONG)	lWidth	(page width (w), see below)
(LONG)	lHeight	(page height (h), see below)
(ULONG)	pInstance	(pointer to instance data)
(ULONG)	lFunction	(High-order word = Flags, Low-order word = NGreSetPageUnits)

ulUnits page units, as:

PU_ARBITRARY Isotropic, arbitrary units, defined by lHeight and lWidth. The page viewport is constructed to give equal x and y spacing on the physical device with at least one dimension of the page completely filling the corresponding default device dimension (maximized window size, paper size). The origin is at the bottom left.

PU_PELS Pel coordinates, with the origin at the bottom left.

PU_LOMETRIC Low resolution metric. These are units of 0.1 mm, with the origin at the bottom left.

PU_HIMETRIC High resolution metric. These are units of 0.01 mm, with the origin at the bottom left.

PU_LOENGLISH Units of 0.01 inch, with the origin at the bottom left.

PU_HIENGLISH Units of 0.001 inch, with the origin at the bottom left.

PU_TWIPS Twentieths of an imperial point. These are units of 1/1440 inch, with the origin at the bottom left.

Other bits are reserved and must be preserved and returned by GetPageUnits.

lWidth/lHeight For PU_ARBITARY, when either lWidth or lHeight is passed as zero, that dimension is set to produce equal x and y spacing on the physical device, with both dimensions completely filling the default device dimensions. When both are passed as zero, then they are set to produce equal x and y spacing on the physical device. lWidth and lHeight completely fill the default device dimensions so that one is equal to the corresponding default device dimension and the other is equal to, or greater than, its corresponding default device dimension. These are measured in pels.

For other units, a value of zero for w or h causes the page to be set to the corresponding default dimension (maximized window size and paper size) in page units, or pels for isotropic units.

Return Codes

On completion, the handling routine must return BOOLEAN (fSuccess):

TRUE Successful.

FALSE Error.

When an error is detected, the handling routine must call WinSetErrorInfo to post the condition. Error codes for conditions that the handling routine is expected to check include:

PMERR_BASE_ERROR
PMERR_INV_IN_AREA
PMERR_PATH_LIMIT_EXCEEDED
PMERR_INV_IN_PATH

PMERR_HDC_BUSY
PMERR_DEV_FUNC_NOT_INSTALLED
PMERR_COORDINATE_OVERFLOW
PMERR_INV_MATRIX_ELEMENT
PMERR_INV_PATTERN_REF_PT_ATTR
PMERR_INSUFFICIENT_MEMORY
PMERR_INV_LENGTH_OR_COUNT
PMERR_INV_RECT.

Remarks

This function causes the Window/Viewport Transform, Page Viewport and Device Transform to be updated as follows:

For Pelsup, LoMetric, HiMetric, LoEnglish, HiEnglish, and twips:

Window/Viewport Transform	Unity
Page Viewport	(0.0) (sx*lWidth-1, sy*lHeight-1)
Device Transform	As defined by Page Window, Page Viewport.

Where sx is the horizontal scaling required by page units for the device (or 1 for pelsup) and sy is the vertical scaling required by page units for the device (or 1 for pelsup).

For Isotropic:

Window/Viewport Transform	Unity
Page Viewport	(0.0) (X2,Y2)
Device Transform	As defined by Page Window, Page Viewport.

Where Dh is the default device (maximized window) height in pels, Dw is the default device (maximized window) width in pels. and Par is the pel (width/height) aspect ratio.

X2, Y2 are integers determined as follows:

```
When (lWidth/lHeight) > Par*(Dw/Dh):

      X2=Dw-1
      Y2=Par*Dw*(lWidth/lHeight)-1

When (lWidth/lHeight) < Par*(Dw/Dh):

      X2=(1/Par)*Dh*(lWidth/lHeight)-1
      Y2=Dh-1

Otherwise, when (lWidth/lHeight) = Par*(Dw/Dh):

      X2=Dw-1
      Y2=Dh-1
```

```
(BOOL) GreGetModelXform (hdc, paXformData, pInstance, lFunction)
```

Queries the current model transform matrix. On completion, paXformData is an array of two-dimensional values defining the current model transform matrix.

Support

This function is supported by the graphics engine and can be hooked by the presentation driver.

Stack Frame

(HDC)	hdc	(device-context handle)
(PXFORM)	paXformData	(points to return data in which the array of 6 matrix elements (M11, M12, M21, M22, M41, M42) are to be stored)
(ULONG)	pInstance	(pointer to instance data)
(ULONG)	lFunction	(High-order word = Flags, Low-order word = NGreGetModelXform)

Return Codes

On completion, the handling routine must return BOOLEAN (fSuccess):

TRUE Successful.
FALSE Error.

When an error is detected, the handling routine must call WinSetErrorInfo to post the condition. Error codes for conditions that the handling routine is expected to check include:

PMERR_HDC_BUSY
PMERR_INV_HDC.

GreSetModelXform

```
(BOOL) GreSetModelXform (hdc, paXformData, flOptions, pInstance, lFunction)
```

Sets the model transform matrix elements as specified.

Support

This function is supported by the graphics engine and can be hooked by the presentation driver.

Stack Frame

(HDC)	hdc	(device-context handle)
(PXFORM)	paXformData	(points to an array of 6 matrix elements for two-dimensional formation. These are M11, M12, M21, M22, M41, M42)
(LONG)	flOptions	(specifies how the supplied array should be used to set the matrix, see below)
(ULONG)	pInstance	(pointer to instance data)
(ULONG)	lFunction	(High-order word = Flags, Low-order word = NGreSetModelXform)

flOptions valid values are:

SX_UNITY Set unity transform (ignore array values)
SX_CAT_AFTER Concatenate after
SX_CAT_BEFORE Concatenate before
SX_OVERWRITE Overwrite.

Return Codes

On completion, the handling routine must return BOOLEAN (fSuccess):

TRUE Successful.
FALSE Error.

When an error is detected, the handling routine must call WinSetErrorInfo to post the condition. Error codes for conditions that the handling routine is expected to check include:

PMERR_BASE_ERROR
PMERR_INV_IN_AREA
PMERR_PATH_LIMIT_EXCEEDED
PMERR_INV_IN_PATH
PMERR_HDC_BUSY
PMERR_DEV_FUNC_NOT_INSTALLED
PMERR_COORDINATE_OVERFLOW
PMERR_INV_MATRIX_ELEMENT
PMERR_INV_PATTERN_REF_PT_ATTR
PMERR_INSUFFICIENT_MEMORY
PMERR_INV_LENGTH_OR_COUNT
PMERR_INV_RECT
PMERR_INV_COORDINATE
PMERR_INV_HDC
PMERR_INV_HRGN
PMERR_INV_TRANSFORM_TYPE
PMERR_INV_REGION_CONTROL
PMERR_INV_COORD_SPACE
PMERR_INV_PICK_APERTURE_POSN.

```
(BOOL) GreGetWindowViewportXform (hdc, paXformData, pInstance, lFunction)
```

Queries the current window/viewport transform matrix. On completion, paXformData is an array of two-dimensional values defining the current Window/Viewport transform matrix.

Support

This function is supported by the graphics engine and can be hooked by the presentation driver.

Stack Frame

(HDC)	hdc	(device-context handle)
(PXFORM)	paXformData	(points to return data in which the array of 6 matrix elements (M11, M12, M21, M22, M41, M42) are to be stored)
(ULONG)	pInstance	(pointer to instance data)
(ULONG)	lFunction	(High-order word = Flags, Low-order word = NGreGetWindowViewportXform)

Return Codes

On completion, the handling routine must return BOOLEAN (fSuccess):

TRUE Successful.
FALSE Error.

When an error is detected, the handling routine must call WinSetErrorInfo to post the condition. Error codes for conditions that the handling routine is expected to check include:

PMERR_HDC_BUSY
PMERR_INV_HDC.

GreSetWindowViewportXform

```
(BOOL) GreSetWindowViewportXform (hdc, paXformData, flOptions, pInstance, lFunction)
```

Sets the Window/Viewport transform matrix elements as specified.

Support

This function is supported by the graphics engine and can be hooked by the presentation driver.

Stack Frame

(HDC)	hdc	(device-context handle)
(PXFORM)	paXformData	(points to an array of 6 matrix elements for two-dimensional formation. These are M11, M12, M21, M22, M41, M42)
(LONG)	flOptions	(specifies how the supplied array should be used to set the matrix, see below)
(ULONG)	pInstance	(pointer to instance data)
(ULONG)	lFunction	(High-order word = Flags, Low-order word = NGreSetWindowViewportXform)

flOptions valid values are:

SX_UNITY Set unity transform (ignore array values)
SX_CAT_AFTER Concatenate after
SX_CAT_BEFORE Concatenate before
SX_OVERWRITE Overwrite.

Return Codes

On completion, the handling routine must return BOOLEAN (fSuccess):

TRUE Successful.
FALSE Error.

When an error is detected, the handling routine must call WinSetErrorInfo to post the condition. Error codes for conditions that the handling routine is expected to check include:

PMERR_BASE_ERROR
PMERR_INV_IN_AREA
PMERR_PATH_LIMIT_EXCEEDED
PMERR_INV_IN_PATH
PMERR_HDC_BUSY
PMERR_DEV_FUNC_NOT_INSTALLED
PMERR_COORDINATE_OVERFLOW
PMERR_INV_MATRIX_ELEMENT
PMERR_INV_PATTERN_REF_PT_ATTR
PMERR_INSUFFICIENT_MEMORY
PMERR_INV_LENGTH_OR_COUNT
PMERR_INV_RECT
PMERR_INV_COORDINATE
PMERR_INV_HDC
PMERR_INV_HRGN
PMERR_INV_TRANSFORM_TYPE
PMERR_INV_REGION_CONTROL
PMERR_INV_COORD_SPACE
PMERR_INV_PICK_APERTURE_POSN.

```
(BOOL) GreGetGlobalViewingXform (hdc, paXformData, pInstance, lFunction)
```

Queries the global viewing transform matrix. On completion, paXformData points to an array of two-dimensional values that defines the global viewing transform matrix.

Support

This function is supported by the graphics engine and can be hooked by the presentation driver.

Stack Frame

(HDC)	hdc	(device-context handle)
(PXFORM)	paXformData	(points to return data in which the array of 6 matrix elements (M11, M12, M21, M22, M41, M42) are to be stored)
(ULONG)	pInstance	(pointer to instance data)
(ULONG)	lFunction	(High-order word = Flags, Low-order word = NGreGetGlobalViewingXform)

Return Codes

On completion, the handling routine must return BOOLEAN (fSuccess):

TRUE Successful.
FALSE Error.

When an error is detected, the handling routine must call WinSetErrorInfo to post the condition. Error codes for conditions that the handling routine is expected to check include:

PMERR_HDC_BUSY
PMERR_INV_HDC.

GreSetGlobalViewingXform

```
(BOOL) GreSetGlobalViewingXform (hdc, paXformData, flOptions, pInstance, lFunction)
```

Sets the global viewing transform matrix elements.

Support

This function is supported by the graphics engine and can be hooked by the presentation driver.

Stack Frame

(HDC)	hdc	(device-context handle)
(PXFORM)	paXformData	(points to an array of 6 matrix elements for two-dimensional formation. These are M11, M12, M21, M22, M41, M42)
(LONG)	flOptions	(specifies how the supplied array should be used to set the matrix, see below)
(ULONG)	pInstance	(pointer to instance data)
(ULONG)	lFunction	(High-order word = Flags, Low-order word = NGreSetGlobalViewingXform)

flOptions valid values are:

SX_UNITY Set unity transform (ignore array values)
SX_CAT_AFTER Concatenate after
SX_CAT_BEFORE Concatenate before
SX_OVERWRITE Overwrite.

Return Codes

On completion, the handling routine must return BOOLEAN (fSuccess):

TRUE Successful.
FALSE Error.

When an error is detected, the handling routine must call WinSetErrorInfo to post the condition. Error codes for conditions that the handling routine is expected to check include:

PMERR_BASE_ERROR
PMERR_INV_IN_AREA
PMERR_PATH_LIMIT_EXCEEDED
PMERR_INV_IN_PATH
PMERR_HDC_BUSY
PMERR_DEV_FUNC_NOT_INSTALLED
PMERR_COORDINATE_OVERFLOW
PMERR_INV_MATRIX_ELEMENT
PMERR_INV_PATTERN_REF_PT_ATTR
PMERR_INSUFFICIENT_MEMORY
PMERR_INV_LENGTH_OR_COUNT
PMERR_INV_RECT
PMERR_INV_COORDINATE
PMERR_INV_HDC
PMERR_INV_HRGN
PMERR_INV_TRANSFORM_TYPE
PMERR_INV_REGION_CONTROL
PMERR_INV_COORD_SPACE
PMERR_INV_PICK_APERTURE_POSN.

```
(ULONG) GreSaveXformData (hdc, ulSize, pBuffer, pInstance, lFunction)
```

Stores the current transform state in pBuffer.

Support

This function is supported by the graphics engine and can be hooked by the presentation driver.

Stack Frame

(HDC)	hdc	(device-context handle)
(ULONG)	ulSize	(size, in bytes, required for pBuffer, see below)
(PBYTE)	pBuffer	(pointer to an area where the data is to be stored)
(ULONG)	pInstance	(pointer to instance data)
(ULONG)	lFunction	(High-order word = Flags, Low-order word = NGreSaveXformData)

ulSize this can be specified as zero, in this instance the function returns the size of the transform data:

```
ulSize=GreSaveXformData (hdc, 0, 0);     /* Find out how big the buffer must be */
GreSaveXformData(hdc, ulSize, pBuffer);  /* then save the transform state */
```

Return Codes

When called with ulSize specified as zero, this function returns the size of the buffer required to save the transform state. Otherwise it returns GPI_OK to indicate a successful completion. In either instance, GPI_ERROR is returned to indicate failure.

When an error is detected, the handling routine must call WinSetErrorInfo to post the condition. Error codes for conditions that the handling routine is expected to check include:

PMERR_HDC_BUSY
PMERR_INV_LENGTH_OR_COUNT
PMERR_INV_HDC.

GreRestoreXformData

```
(BOOL) GreRestoreXformData (hdc, ulSize, pBuffer, pInstance, lFunction)
```

Restores a previously saved transform state. The current transform state is overwritten.

Support

This function is supported by the graphics engine and can be hooked by the presentation driver.

Stack Frame

(HDC)	hdc	(device-context handle)
(ULONG)	ulSize	(size, in bytes, of pBuffer)
(PBYTE)	pBuffer	(pointer to stored transform data)
(ULONG)	pInstance	(pointer to instance data)
(ULONG)	lFunction	(High-order word = Flags, Low-order word = NGreRestoreXformData)

Return Codes

This function returns BOOLEAN (fSuccess):

TRUE Successful.
FALSE Error.

When an error is detected, the handling routine must call WinSetErrorInfo to post the condition. Error codes for conditions that the handling routine is expected to check include:

PMERR_BASE_ERROR
PMERR_INV_IN_AREA
PMERR_PATH_LIMIT_EXCEEDED
PMERR_INV_IN_PATH
PMERR_HDC_BUSY
PMERR_DEV_FUNC_NOT_INSTALLED
PMERR_COORDINATE_OVERFLOW
PMERR_INV_PATTERN_REF_PT_ATTR
PMERR_INSUFFICIENT_MEMORY
PMERR_INV_LENGTH_OR_COUNT
PMERR_INV_HDC
PMERR_INV_COORD_SPACE
PMERR_INV_PICK_APERTURE_POSN.

```
(BOOL) GreGetPageViewport (hdc, prcViewport, pInstance, lFunction)
```

Loads the buffer indicated by prcViewport with the page viewport coordinates.

Support

This function is supported by the graphics engine and can be hooked by the presentation driver.

Stack Frame

(HDC)	hdc	(device-context handle)
(PRECTL)	prcViewport	(pointer to page viewport boundaries, see below)
(ULONG)	pInstance	(pointer to instance data)
(ULONG)	lFunction	(High-order word = Flags, Low-order word = NGreGetPageViewport)

prcViewport This is a RECTL structure in device coordinates:

xLeft Minimum x coordinate of viewport
yBottom Minimum y coordinate
xRight Maximum x coordinate of viewport
yTop Maximum y coordinate.

Boundaries are inclusive at the bottom-left corner and exclusive at the top right.

Return Codes

On completion, the handling routine must return BOOLEAN (fSuccess):

TRUE Successful.
FALSE Error.

When an error is detected, the handling routine must call WinSetErrorInfo to post the condition. Error codes for conditions that the handling routine is expected to check include:

PMERR_HDC_BUSY
PMERR_INV_HDC.

GreSetPageViewport

```
(BOOL) GreSetPageViewport (hdc, prcViewport, flOptions, pInstance, lFunction)
```

Sets the page viewport in device coordinates so that the engine can update the device transform using the page window and page viewport coordinates.

Support

This function is supported by the graphics engine and can be hooked by the presentation driver.

Stack Frame

(HDC)	hdc	(device-context handle)
(PRECTL)	prcViewport	(pointer to page viewport boundaries, in device coordinates see below)
(ULONG)	flOptions	(this is a reserved parameter, the only valid value is zero)
(ULONG)	pInstance	(pointer to instance data)
(ULONG)	lFunction	(High-order word = Flags, Low-order word = NGreSetPageViewport)

prcViewport RECTL structure:

xLeft Minimum x coordinate of viewport
yBottom Minimum y coordinate
xRight Maximum x coordinate of viewport
yTop Maximum y coordinate.

Return Codes

On completion, the handling routine must return BOOLEAN (fSuccess):

TRUE Successful.
FALSE Error.

When an error is detected, the handling routine must call WinSetErrorInfo to post the condition. Error codes for conditions that the handling routine is expected to check include:

PMERR_BASE_ERROR
PMERR_INV_IN_AREA
PMERR_PATH_LIMIT_EXCEEDED
PMERR_INV_IN_PATH
PMERR_HDC_BUSY
PMERR_DEV_FUNC_NOT_INSTALLED
PMERR_COORDINATE_OVERFLOW
PMERR_INV_MATRIX_ELEMENT
PMERR_INV_PATTERN_REF_PT_ATTR
PMERR_INSUFFICIENT_MEMORY
PMERR_INV_LENGTH_OR_COUNT
PMERR_INV_RECT
PMERR_INV_COORDINATE
PMERR_INV_HDC
PMERR_INV_HRGN
PMERR_INV_SET_VIEWPORT_OPTION
PMERR_INV_PAGE_VIEWPORT
PMERR_INV_REGION_CONTROL
PMERR_INV_COORD_SPACE
PMERR_INV_PICK_APERTURE_POSN.

```
(BOOL) GreGetGraphicsField (hdc, prcGraphicsField, pInstance, lFunction)
```

Loads the buffer indicated by prcGraphicsField with the integer values that identify the boundaries of the graphics field.

Support

This function is supported by the graphics engine and can be hooked by the presentation driver.

Stack Frame

(HDC)	hdc	(device-context handle)
(PRECTL)	prcGraphicsField	(pointer to graphics field, see below)
(ULONG)	pInstance	(pointer to instance data)
(ULONG)	lFunction	(High-order word = Flags, Low-order word = NGreGetGraphicsField)

prcGraphicsField RECTL structure:

xLeft	Minimum x coordinate of graphics field
yBottom	Minimum y coordinate
xRight	Maximum x coordinate of graphics field
yTop	Maximum y coordinate.

Return Codes

On completion, the handling routine must return BOOLEAN (fSuccess):

TRUE	Successful.
FALSE	Error.

When an error is detected, the handling routine must call WinSetErrorInfo to post the condition. Error codes for conditions that the handling routine is expected to check include:

PMERR_HDC_BUSY
PMERR_INV_HDC.

GreSetGraphicsField

```
(BOOL) GreSetGraphicsField (hdc, prcGraphicsField, pInstance, lFunction)
```

Sets the boundaries of the graphics field (clip) limits in page coordinate space.

The boundaries are inclusive, so that points on them are not clipped. By default, no clipping is performed.

Support

This function is supported by the graphics engine and can be hooked by the presentation driver.

Stack Frame

(HDC)	hdc	(device-context handle)
(PRECTL)	prcGraphicsField	(pointer to graphics field, see below)
(ULONG)	pInstance	(pointer to instance data)
(ULONG)	lFunction	(High-order word = Flags, Low-order word = NGreSetGraphicsField)

prcGraphicsField RECTL structure:

xLeft Minimum x coordinate of graphics field
yBottom Minimum y coordinate
xRight Maximum x coordinate of graphics field
yTop Maximum y coordinate.

An error is raised when xLeft is greater than xRight or yBottom is greater than yTop.

Return Codes

On completion, the handling routine must return BOOLEAN (fSuccess):

TRUE Successful.
FALSE Error.

When an error is detected, the handling routine must call WinSetErrorInfo to post the condition. Error codes for conditions that the handling routine is expected to check include:

PMERR_INV_IN_AREA
PMERR_INV_IN_PATH
PMERR_HDC_BUSY
PMERR_INV_COORDINATE
PMERR_INV_HDC
PMERR_INV_GRAPHICS_FIELD.

```
(BOOL) GreGetViewingLimits (hdc, prcViewingLimits, pInstance, lFunction)
```

Loads the array, indicated by prcViewingLimits, with the boundaries of the viewing window in graphic model space coordinates. All boundaries are inclusive.

Support

This function is supported by the graphics engine and can be hooked by the presentation driver.

Stack Frame

(HDC)	hdc	(device-context handle)
(PRECTL)	prcViewingLimits	(pointer to limits of viewing area, see below)
(ULONG)	pInstance	(pointer to instance data)
(ULONG)	lFunction	(High-order word = Flags, Low-order word = NGreGetViewingLimits)

prcViewingLimits RECTL structure:

xLeft	Minimum x coordinate of viewing limits
yBottom	Minimum y coordinate
xRight	Maximum x coordinate of viewing limits
yTop	Maximum y coordinate.

Return Codes

On completion, the handling routine must return BOOLEAN (fSuccess):

TRUE	Successful.
FALSE	Error.

When an error is detected, the handling routine must call WinSetErrorInfo to post the condition. Error codes for conditions that the handling routine is expected to check include:

PMERR_HDC_BUSY
PMERR_INV_HDC.

GreSetViewingLimits

```
(BOOL) GreSetViewingLimits (hdc, prcViewingLimits, pInstance, lFunction)
```

Sets the boundaries of the viewing (clip) limits, in model space, to the specified values.

The boundaries are inclusive and are not clipped.

The viewing-limits coordinates are transformed to make a clipping rectangle in page, or device, coordinate space. Any rotation or shear of this rectangle is ignored. When either the left boundary is greater than the right, or the bottom boundary is greater than the top, a null rectangle is defined and all points are clipped.

Support

This function is supported by the graphics engine and can be hooked by the presentation driver.

Stack Frame

(HDC)	hdc	(device-context handle)
(PRECTL)	prcViewLimits	(pointer to limits of viewing area, see below)
(ULONG)	pInstance	(pointer to instance data)
(ULONG)	lFunction	(High-order word = Flags, Low-order word = NGreSetViewingLimits)

prcViewLimits RECTL structure:

xLeft Minimum x coordinate of viewing limits
yBottom Minimum y coordinate
xRight Maximum x coordinate of viewing limits
yTop Maximum y coordinate.

Return Codes

On return, the handling routine must return BOOLEAN (fSuccess):

TRUE Successful.
FALSE Error.

When an error is detected, the handling routine must call WinSetErrorInfo to post the condition. Error codes for conditions that the handling routine is expected to check include:

PMERR_INV_IN_AREA
PMERR_INV_IN_PATH
PMERR_HDC_BUSY
PMERR_INV_COORDINATE
PMERR_INV_HDC
PMERR_INV_GRAPHICS_FIELD.

```
(BOOL) GreConvert (hdc, lSrc, lDst, paptPoints, cPoints, pInstance, lFunction)
```

Converts the specified coordinates from one coordinate space to another, using the current values of the transforms.

Support

This function is supported by the graphics engine and can be hooked by the presentation driver.

Stack Frame

(HDC)	hdc	(device-context handle)
(LONG)	lSrc	(source coordinate space, see below)
(LONG)	lDst	(target coordinate space, see below)
(PPOINTL)	paptPoints	(long pointer to array of (x,y) coordinates to transform, the result is also returned to this parameter)
(LONG)	cPoints	(count of coordinate pairs in the array)
(ULONG)	pInstance	(pointer to instance data)
(ULONG)	lFunction	(High-order word = Flags, Low-order word = NGreConvert)

lSrc, lDst These define the source and target coordinate space:

CVTC_WORLD World coordinate space
CVTC_MODEL Model space
CVTC_DEFAULTPAGE Default page coordinate space
CVTC_PAGE Page coordinate space
CVTC_DEVICE Device coordinate space. Screen coordinates are 32-bit signed integers, and are used by the presentation driver as screen pel addresses.

Return Codes

On completion, the handling routine must return BOOLEAN (fSuccess):

TRUE Successful.
FALSE Error.

When an error is detected, the handling routine must call WinSetErrorInfo to post the condition. Error codes for conditions that the handling routine is expected to check include:

PMERR_HDC_BUSY
PMERR_COORDINATE_OVERFLOW
PMERR_INV_LENGTH_OR_COUNT
PMERR_INV_HDC
PMERR_INV_COORD_SPACE.

GreSaveXform

```
(BOOL) GreSaveXform (hdc, cSave, pInstance, lFunction)
```

This function is called during SaveDC and OpenDC to allow simulations to save their local data structures.

When a new DC is created, this function is called with a count of 1 to initialize its local data.

A save with a count of one more than the current save level generates intervening levels.

Support

This function is supported by the graphics engine and can be hooked by the presentation driver.

Stack Frame

(HDC)	hdc	(device-context handle)
(LONG)	cSave	(DC save level, this must not be a negative value)
(ULONG)	pInstance	(pointer to instance data)
(ULONG)	lFunction	(High-order word = Flags, Low-order word = NGreSaveXform)

Return Codes

On completion, the handling routine must return BOOLEAN (fSuccess):

TRUE Successful.
FALSE Error.

When an error is detected, the handling routine must call WinSetErrorInfo to post the condition. Error codes for conditions that the handling routine is expected to check include:

PMERR_HDC_BUSY
PMERR_INSUFFICIENT_MEMORY
PMERR_INV_HDC.

Remarks

When initializing a new DC, the default transform must be set.

```
(BOOL) GreRestoreXform (hdc, cSave, pInstance, lFunction)
```

This function is called during RestoreDC and CloseDC to allow simulations to restore their local data structures.

When a DC is closed, this function is called with a count of zero to free its local data.

Support

This function is supported by the graphics engine and can be hooked by the presentation driver.

Stack Frame

(HDC)	hdc	(device-context handle)
(LONG)	cSave	(DC save level, see below)
(ULONG)	pInstance	(pointer to instance data)
(ULONG)	lFunction	(High-order word = Flags, Low-order word = NGreRestoreXform)

cSave Indicates the saved DC state that the handling routine should use to restore the transform.

When cSave is passed as −1, the handling routine resets the transform to its initial state (−1 is passed to this routine from GreResetDC and is the only valid negative value). A value of zero also indicates that the transform should be reset to its initial state; other positive values identify which saved level should be restored (see "Enable subfunction 08H − RestoreDCState" on page 11-16).

Return Codes

On completion, the handling routine must return BOOLEAN (fSuccess):

TRUE Successful.
FALSE Error.

When an error is detected, the handling routine must call WinSetErrorInfo to post the condition. Error codes for conditions that the handling routine is expected to check include:

PMERR_HDC_BUSY
PMERR_INV_HDC.

GreMultiplyXforms

```
(BOOL) GreMultiplyXforms (hdc, paXform, paNewXformData, lMode, pInstance, lFunction)
```

Multiplies the transform matrix, defined by paNewXformData, by the corresponding matrix in paXform. The result is stored in paXform.

When this function is used to make a series of matrix multiplications on the same matrix, some loss of accuracy can occur due to rounding, as no higher precision can be retained across calls.

Support

This function is supported by the graphics engine and can be hooked by the presentation driver.

Stack Frame

(HDC)	hdc	(device-context handle)
(PXFORM)	paXform	(points to an array of 6 matrix elements for two-dimensional formation. These are M11, M12, M21, M22, M41, M42)
(PXFORM)	paNewXformData	(points to an array of 6 matrix elements for two-dimensional formation. These are M11, M12, M21, M22, M41, M42)
(LONG)	lMode	(specifies how the supplied array should be used to set the matrix, see below)
(ULONG)	pInstance	(pointer to instance data)
(ULONG)	lFunction	(High-order word = Flags, Low-order word = NGreMultiplyXforms)

lMode

SX_UNITY	Set unity transform (ignore array values)
SX_CAT_AFTER	Concatenate after
SX_CAT_BEFORE	Concatenate before
SX_OVERWRITE	Overwrite.

Return Codes

On completion, the handling routine must return BOOLEAN (fSuccess):

TRUE	Successful.
FALSE	Error.

When an error is detected, the handling routine must call WinSetErrorInfo to post the condition. Error codes for conditions that the handling routine is expected to check include:

PMERR_INV_MATRIX_ELEMENT
PMERR_INV_TRANSFORM_TYPE.

```
(BOOL) GreConvertWithMatrix (hdc, paptPoints, cPoints, paXform, pInstance, ulFunction)
```

Converts a series of points using the matrix indicated, other current transform matrixes are ignored.

Support

This function is supported by the graphics engine and can be hooked by the presentation driver.

Stack Frame

(HDC)	hdc	(device-context handle)
(PPOINTL)	paptPoints	(long pointer to array of (x,y) coordinates to transform, the result is also returned to this parameter)
(LONG)	cPoints	(count of coordinate pairs in the array)
(PXFORM)	paXform	(points to an array of 6 matrix elements for two-dimensional formation. These are M11, M12, M21, M22, M41, M42)
(ULONG)	pInstance	(pointer to instance data)
(ULONG)	lFunction	(High-order word = Flags, Low-order word = NGreConvertWithMatrix)

Return Codes

On completion, the handling routine must return BOOLEAN (fSuccess):

TRUE Successful.
FALSE Error.

When an error is detected, the handling routine must call WinSetErrorInfo to post the condition. Error codes for conditions that the handling routine is expected to check include:

PMERR_COORDINATE_OVERFLOW
PMERR_INV_LENGTH_OR_COUNT.

Chapter 15. Graphics Engine Internal Functions

The handling routines for these functions are in the graphics engine and they are called through InnerGreEntry. Because they are not called through the dispatch table, they cannot be hooked by the presentation driver.

For a description of InnerGreEntry, see page 16-8.

Calls described in this chapter are in the following function groups:

- Device context functions (INCL_GRE_DCS)
- Device support functions (INCL_GRE_DEVSUPPORT)
- SetID functions (INCL_GRE_SETID)
- LCID functions (INCL_GRE_LCID)
- Font functions (INCL_GRE_FONTS)
- Journal functions (INCL_GRE_JOURNALING).

Each description shows what the function does, the parameters passed to the engine and the values that the function returns. These functions are grouped according to the conditional include sections of the header file PMDDIM.H.

Note: The Gre... macros defined in PMDDIM.H must not be used to call to these functions from ring 2.

Device Context Functions

- GreOpenDC
- GreCloseDC
- GreResetDC
- GreGetHandle
- GreSetHandle
- GreGetProcessControl
- GreSetProcessControl
- GreSaveDC
- GreRestoreDC
- GreQueryDeviceNames
- GreQueryEngineVersion.

```
(HDC) GreOpenDC (hdc, lType, pszToken, cData, pszData, pInstance, lFunction)
```

Creates an output device context (DC).

The new device context inherits the current code page of the process that created it. Subsequent calls to DosSetCp do not alter the code page of an existing DC. Default VIO and KBD code pages are always in the last code page set by any application process.

Support

This function is supported by the graphics engine. It is available for use by the presentation driver at InnerGreEntry.

Stack Frame

(HDC)	hdc	(device-context handle, see below)
(LONG)	lType	(type of DC being opened, see below)
(PSZ)	pszToken	(this parameter is ignored by the graphics engine)
(LONG)	cData	(number of elements in the data structure)
(PDEVOPENDATA)	pszData	(pointer to data structure, see below)
(ULONG)	pInstance	(pointer to instance data)
(ULONG)	lFunction	(High-order word = Flags, Low-order word = NGreOpenDC)

hdc When lType is OD_MEMORY, this parameter is a handle to a device context compatible with bit maps that are used with this device context. If this is not supplied, or is NULL, compatibility with the screen is assumed.

lType Type of device context:

- **OD_QUEUED** The DC is for a device such as a printer or plotter, for which output is to be queued by the spooler.
- **OD_DIRECT** The DC is for a device, for which output is not to be queued by the spooler.
- **OD_INFO** This is similar to OD_DIRECT, except that it is only used to retrieve information, such as font metrics. Drawing can be done to a presentation space associated with such a DC, but no output medium is updated.
- **OD_MEMORY** The new DC is a memory DC, used to contain a bit map. hdc identifies the device with which the bit map is to be compatible.

pszData Long pointer to DEVOPENSTRUC structure:

- **pszLogAddress** Pointer to logical address. (For example, 'LPT1'.)
- **pszDriverName** Pointer to presentation driver name. (For example, 'IBM4201'.)
- **pdriv** Pointer to a DRIVDATA structure:
 - **cb** Size, in bytes, of this structure.
 - **lVersion** Version number of the data. Version numbers are defined by the presentation driver.
 - **szDeviceName[32]** String identifying the device. Valid values are supplied by the presentation driver.
 - **abGeneralData** General data which is defined by the presentation driver. This must not contain pointers, as these are not necessarily valid when passed to the driver.
- **pszDataType** Data type of the queued file, supported data types are:
 PM_Q_STD
 PM_Q_RAW.
 User-defined data types can also be supported.
- **pszComment** Pointer to a description of the file which could, for example, be displayed by the spooler to the user.

pszQueueProcName Pointer to name of queue processor.

pszQueueProcParams Pointer to a string of queue processor parameters.

pszSpoolerParams Pointer to a string of spooler parameters separated by one or more blanks. Valid parameters are:

FORM = aaa

Identifies the form name for a print job; multiple names should be separated by commas (aaa,bbb,ccc). If this parameter is not present, the current form is used.

Form names are defined by the presentation driver; valid names are those that would be returned from a call to the driver's GreQueryHardcopyCaps handling routine.

PRTY = n

Identifies the priority for the print job. The priority can be any value from 1 through 99 (1 is lowest priority). If this parameter is not present, the priority value defaults to 50.

pszNetworkParams Pointer to string of networking parameters. These are only used in a network environment and their nature is defined by the network application.

ulReserved Reserved.

Return Codes

On completion, the graphics engine returns the handle of the new device context (hdc) or DEV_ERROR if an error occurred.

When an error is detected, the graphics engine calls WinSetErrorInfo to post the condition. Reasons for failure of this function include:

PMERR_BASE_ERROR
PMERR_INV_IN_AREA
PMERR_INV_IN_PATH
PMERR_HDC_BUSY
PMERR_DEV_FUNC_NOT_INSTALLED
PMERR_COORDINATE_OVERFLOW
PMERR_INV_PATTERN_SET_ATTR
PMERR_INV_MIX_ATTR
PMERR_INV_BACKGROUND_MIX_ATTR
PMERR_INV_PATTERN_REF_PT_ATTR
PMERR_INV_LINE_TYPE_ATTR
PMERR_INV_CHAR_MODE_ATTR
PMERR_INV_CHAR_DIRECTION_ATTR
PMERR_INSUFFICIENT_MEMORY
PMERR_INV_LENGTH_OR_COUNT
PMERR_INV_CODEPAGE
PMERR_EXCEEDS_MAX_SEG_LENGTH
PMERR_INV_RECT
PMERR_INV_COORDINATE
PMERR_INV_HDC
PMERR_INV_HRGN
PMERR_INV_DC_DATA
PMERR_INV_REGION_CONTROL
PMERR_INV_DRIVER_NAME
PMERR_INV_HBITMAP
PMERR_INV_DC_TYPE
PMERR_INV_COORD_SPACE
PMERR_INV_ID
PMERR_BITMAP_IS_SELECTED
PMERR_HBITMAP_BUSY
PMERR_INV_INFO_TABLE
PMERR_INV_BITMAP_DIMENSION

PMERR_INV_SCAN_START
PMERR_INV_PATTERN_SET_FONT
PMERR_HUGE_FONTS_NOT_SUPPORTED
PMERR_INV_COLOR_ATTR
PMERR_INV_BACKGROUND_COL_ATTR.

Remarks

When this function is called for the first time, the graphics engine performs the following sequence:

```
DosLoadModule("NEWDRIVER", &hDriver)
                /*****************************************************/
                /*  Load the presentation driver .DLL file.          */
                /*****************************************************/
DosGetProcAddr(hDriver, "OS2_PM_DRV_ENABLE", &Enable)
                /*****************************************************/
                /*  Find the presentation driver's Enable       */
                /*  function.                                        */
                /*  See "OS2_PM_DRV_ENABLE" on page 11-5.                    */
                /*****************************************************/
*Enable (FillLogicalDeviceBlock, &DispatchTable)
                /*****************************************************/
                /*  The presentation driver must:              */
                /*  Save the addresses of the engine simulations.   */
                /*  Overwrite the dispatch table, as necessary.     */
                /*  Hook the ExitList for the calling process.      */
                /*****************************************************/
hPhysDev=*Enable (FillPhysicalDeviceBlock, pDevOpenStructure)
                /*****************************************************/
                /*  Create the physical device block.                */
                /*****************************************************/
pInstance=*Enable (EnableDeviceContext, hdc, Type, hPhysDev)
                /*****************************************************/
                /* The presentation driver must create an instance  */
                /* data structure for the DC.                       */
                /*****************************************************/
*Enable(CompleteOpenDC, hdc, pInstance)
                /*****************************************************/
                /* The presentation driver must inform the system   */
                /* that the DC is open and ready to receive output. */
                /*****************************************************/
```

For subsequent calls to GreOpenDc, for the same DC, the graphics engine calls Enable(EnableDeviceContext) to create a new instance for the DC and then calls Enable(CompleteOpenDC).

When this function is called for the same DC by a different process, the graphics engine calls DosLoadModule to load the presentation driver. It then calls Enable(FillLogicalDeviceBlock), Enable(EnableDeviceContext) and Enable(CompleteOpenDC).

GreCloseDC

```
(BOOL) GreCloseDC (hdc, pInstance, lFunction)
```

Closes a device context.

If the DC is a memory DC with a bit map selected, the engine deselects the bit map before destroying the DC. This function deletes any visible regions and clip regions selected into the DC.

Support

This function is supported by the graphics engine. It is available for use by the presentation driver at InnerGreEntry.

Stack Frame

(HDC)	hdc	(device-context handle)
(ULONG)	pInstance	(pointer to instance data)
(ULONG)	lFunction	(High-order word = Flags, Low-order word = NGreCloseDC)

Return Codes

On completion, the handling routine must return BOOLEAN (fSuccess):

TRUE Successful.
FALSE Error.

When an error is detected, the graphics engine calls WinSetErrorInfo to post the condition. Reasons for failure of this function include:

PMERR_BASE_ERROR
PMERR_INV_IN_AREA
PMERR_INV_IN_PATH
PMERR_HDC_BUSY
PMERR_DEV_FUNC_NOT_INSTALLED
PMERR_INSUFFICIENT_MEMORY
PMERR_INV_LENGTH_OR_COUNT
PMERR_INV_CODEPAGE
PMERR_EXCEEDS_MAX_SEG_LENGTH
PMERR_INV_RECT
PMERR_INV_COORDINATE
PMERR_INV_HDC
PMERR_INV_HRGN
PMERR_INV_REGION_CONTROL
PMERR_INV_HBITMAP
PMERR_INV_DC_TYPE
PMERR_INV_ID
PMERR_BITMAP_IS_SELECTED
PMERR_HBITMAP_BUSY
PMERR_INV_INFO_TABLE
PMERR_INV_BITMAP_DIMENSION
PMERR_INV_SCAN_START.

```
(BOOL) GreResetDC (hdc, flOptions, pInstance, lFunction)
```

Calling this function restores a DC to its created state. All objects, such as fonts, patterns, and paths, are deleted and all attributes are set to their defaults; any clip region selected into the DC is deleted.

This function does not alter window-manager information stored in the DC instance data. Window-manager information includes:

- The visible region
- The DC origin
- User bounds
- Cached clipping rectangles
- The HDC_IS_DIRTY flag.

Support

This function is supported by the graphics engine. It is available for use by the presentation driver at InnerGreEntry.

Stack Frame

(HDC)	hdc	(device-context handle)
(ULONG)	flOptions	(option flags, see below)
(ULONG)	pInstance	(pointer to instance data)
(ULONG)	lFunction	(High-order word = Flags, Low-order word = NGreResetDC)

flOptions Valid flags are:

RDC_SETOWNERTOSHELL When this flag is set, the engine sets the ownership of the reset DC to the process that first initialized the graphics engine. This is normally the Presentation Manager.

RDC_RGBMODE Sets the DC's logical color table to RGB mode.

Return Codes

On completion, the handling routine must return BOOLEAN (fSuccess):

TRUE Successful.
FALSE Error.

When an error is detected, the graphics engine calls WinSetErrorInfo to post the condition. Reasons for failure of this function include:

PMERR_BASE_ERROR
PMERR_INV_IN_AREA
PMERR_INV_IN_PATH
PMERR_HDC_BUSY
PMERR_DEV_FUNC_NOT_INSTALLED
PMERR_INSUFFICIENT_MEMORY
PMERR_INV_LENGTH_OR_COUNT
PMERR_INV_CODEPAGE
PMERR_EXCEEDS_MAX_SEG_LENGTH
PMERR_INV_RECT
PMERR_INV_COORDINATE
PMERR_INV_HDC
PMERR_INV_HRGN
PMERR_INV_REGION_CONTROL
PMERR_INV_HBITMAP
PMERR_INV_DC_TYPE
PMERR_INV_ID

PMERR_BITMAP_IS_SELECTED
PMERR_HBITMAP_BUSY
PMERR_INV_INFO_TABLE
PMERR_INV_BITMAP_DIMENSION
PMERR_INV_SCAN_START.

```
(LONG) GreGetHandle (hdc, iIndex, pInstance, lFunction)
```

Returns the handle, or variable stored in the DC corresponding to iIndex, previously set by GreSetHandle.

Support

This function is supported by the graphics engine. It is available for use by the presentation driver at InnerGreEntry.

Stack Frame

(HDC)	hdc	(device context handle)
(ULONG)	iIndex	(index value of the returned handle, in the range 0 through 3)
(ULONG)	pInstance	(pointer to instance data)
(ULONG)	lFunction	(High-order word = Flags, Low-order word = NGreGetHandle)

iIndex A value of 1 can be used to get the associated AVIO presentation-space handle. See "VioGetPSAddress" on page 16-13.

Return Codes

On completion, the graphics engine returns the handle requested (hHandle) or GPI_ALTERROR, if an error occurred.

When an error is detected, the graphics engine calls WinSetErrorInfo to post the condition. Reasons for failure of this function include:

PMERR_HDC_BUSY
PMERR_INV_HDC.

GreSetHandle

```
(BOOL) GreSetHandle (hdc, hHandle, iIndex, pInstance, lFunction)
```

Stores a handle, or variable, in the device context. Up to four handles can be stored in a DC at one time.

The presentation driver can only use this function for device contexts that it has created, for its own use, with GreOpenDC.

Support

This function is supported by the graphics engine. It is available for use by the presentation driver at InnerGreEntry.

Stack Frame

(ULONG)	hdc	(device context handle)
(ULONG)	hHandle	(Handle to be associated with hdc)
(ULONG)	iIndex	(index value of the handle, in the range 0 through 3, see below)
(ULONG)	pInstance	(pointer to instance data)
(ULONG)	lFunction	(High-order word = Flags, Low-order word = NGreSetHandle)

iIndex For a normal device context, this is a reserved parameter.

Return Codes

On completion, the handling routine must return BOOLEAN (fSuccess):

TRUE Successful.
FALSE Error.

When an error is detected, the graphics engine calls WinSetErrorInfo to post the condition. Reasons for failure of this function include:

PMERR_HDC_BUSY
PMERR_INV_HDC.

```
(LONG) GreGetProcessControl (hdc, pInstance, lFunction)
```

Returns the process control flags.

Support

This function is supported by the graphics engine. It is available for use by the presentation driver at InnerGreEntry.

Stack Frame

(HDC)	hdc	(device-context handle)
(ULONG)	pInstance	(pointer to instance data)
(ULONG)	lFunction	(High-order word = Flags, Low-order word = NGreGetProcessControl)

Return Codes

This function returns the process control flags (flProcess) or DCTL_ERROR if an error occurred. The flags returned are:

PCTL_DRAW Draw flag.
PCTL_BOUND GPI_BOUNDS flag.
PCTL_CORRELATE Correlate flag.
PCTL_USERBOUNDS USER_BOUNDS flag.
PCTL_AREA When set, an area definition is in progress.
COM_PATH When set, a path definition is in progress.

Other flags are reserved and should be ignored.

When an error is detected, the graphics engine calls WinSetErrorInfo to post the condition. Reasons for failure of this function include:

PMERR_HDC_BUSY
PMERR_INV_HDC.

GreSetProcessControl

```
(BOOL) GreSetProcessControl (hdc, flMask, flProcess, pInstance, lFunction)
```

This function provides a mechanism to control drawing, boundary computation, and correlation.

Bounds are returned in graphics-model-space coordinates. If a composite transform is applied to the drawing primitives, the bounds values must be transformed back to their original values before merging with the previous bounds values.

Correlation is done in page-coordinate space on the output of primitives that have been clipped to the viewing limits and graphics field only.

Note: Correlation is done for all functions *except* alphanumeric functions and GreErasePS. Boundary computation is done for all functions *except* GreErasePS.

Support

This function is supported by the graphics engine. It is available for use by the presentation driver at InnerGreEntry.

Stack Frame

(HDC)	hdc	(device-context handle)
(ULONG)	flMask	(only those flags with the corresponding bit in this parameter set are modified)
(ULONG)	flProcess	(process flags, see below.
(ULONG)	pInstance	(pointer to instance data)
(ULONG)	lFunction	(High-order word = Flags, Low-order word = NGreSetProcessControl)

flProcess Process-control flags:

PCTL_DRAW	This flag has no effect on GreErasePS. If set, drawing primitives should appear on screen. Otherwise output operations, such as GreBitblt, GrePaintRegion, GreSetPel, and other drawing primitives, are not displayed.
PCTL_BOUND	This flag is set to indicate that GPI_BOUNDS must be accumulated.
PCTL_CORRELATE	This flag is set to indicate that correlation is to be done.
PCTL_USERBOUND	This flag, when set, indicates that USER_BOUND are to be collected for the window manager.

Return Codes

This function returns BOOLEAN (fSuccess):

TRUE	Successful.
FALSE	Error.

When an error is detected, the graphics engine calls WinSetErrorInfo to post the condition. Reasons for failure of this function include:

PMERR_HDC_BUSY
PMERR_INV_HDC.

```
(LONG) GreSaveDC (hdc, pInstance, lFunction)
```

This function saves the device context's state on a stack and returns an identifier to allow for its subsequent restoration.

The following information is saved:

- Current position.
- Current attributes.
- Current transforms, viewing limits, and clip path.
- Any reference to a selected clip window.
- Any loaded logical color table.
- References to any loaded logical fonts.
- References to the regions created on the associated DC.

The following are not saved:

- Visible region.
- Process controls.

Any resources, such as clip region and logical fonts, that are referenced in a saved DC should not be deleted.

The ID of a saved DC is only unique within the DC for which it is issued. Other DCs can have saved states with the same ID.

The returned identifier can be used for a subsequent GreRestoreDC. This identifier represents the level of the saved DC on the save/restore stack. The first DC saved is identified by a value of 1, the second by 2, and so on.

Support

This function is supported by the graphics engine. It is available for use by the presentation driver at InnerGreEntry.

Stack Frame

(ULONG)	hdc	(device context handle)
(ULONG)	pInstance	(pointer to instance data)
(ULONG)	lFunction	(High-order word = Flags, Low-order word = NGreSaveDC)

Return Codes

This function returns the identifier for the saved DC state (idDC) or GPI_ERROR if an error occurred.

When an error is detected, the graphics engine calls WinSetErrorInfo to post the condition. Reasons for failure of this function include:

PMERR_BASE_ERROR
PMERR_INV_IN_AREA
PMERR_INV_IN_PATH
PMERR_HDC_BUSY
PMERR_DEV_FUNC_NOT_INSTALLED
PMERR_INSUFFICIENT_MEMORY
PMERR_INV_CODEPAGE
PMERR_EXCEEDS_MAX_SEG_LENGTH
PMERR_INV_RECT
PMERR_INV_COORDINATE
PMERR_INV_HDC
PMERR_INV_HRGN
PMERR_INV_REGION_CONTROL
PMERR_INV_DC_TYPE.

GreRestoreDC

```
(BOOL) GreRestoreDC (hdc, idDC, pInstance, lFunction)
```

This function restores the contents of a previously saved device context.

Support

GreRestoreDC is supported by the graphics engine. It is available for use by the presentation driver at InnerGreEntry.

Stack Frame

(HDC)	hdc	(device-context handle)
(LONG)	idDC	(DC state identifier)
(ULONG)	pInstance	(pointer to instance data)
(ULONG)	lFunction	(High-order word = Flags, Low-order word = NGreRestoreDC)

idDC This identifies the DC state to be restored. When this is passed as a negative value, it indicates the number of DCs that must be popped off the stack to access the required DC; −1 indicates that the most recently saved DC must be restored.

- When this parameter is passed as a positive value (>0) and a corresponding DC does not exist, an error is returned. The current DC is not modified.
- When this is passed as a negative value and there are insufficient entries on the stack, an error is raised. The current DC is not modified.
- When the value passed corresponds to a saved DC, all entries on the on the stack, up to that indicated, are discarded. Any clip regions selected into the discarded DCs are destroyed.
- A value of 1 resets the DC stack, all entries are removed from the stack.
- A value of 0 raises an error and the DC is not modified.

Return Codes

This function returns BOOLEAN (fSuccess):

TRUE Successful.
FALSE Error.

When an error is detected, the graphics engine calls WinSetErrorInfo to post the condition. Reasons for failure of this function include:

PMERR_BASE_ERROR
PMERR_INV_IN_AREA
PMERR_INV_IN_PATH
PMERR_HDC_BUSY
PMERR_DEV_FUNC_NOT_INSTALLED
PMERR_INSUFFICIENT_MEMORY
PMERR_INV_CODEPAGE
PMERR_EXCEEDS_MAX_SEG_LENGTH
PMERR_INV_RECT
PMERR_INV_COORDINATE
PMERR_INV_HDC
PMERR_INV_HRGN
PMERR_INV_REGION_CONTROL
PMERR_INV_DC_TYPE
PMERR_INV_ID.

```
(LONG) GreQueryEngineVersion (pInstance, lFunction)
```

This function returns the version number of the graphics engine.

Support

GreQueryEngineVersion is supported by the graphics engine. It is available for use by the presentation driver at InnerGreEntry.

Stack Frame

(ULONG)	pInstance	(pointer to instance data)
(ULONG)	lFunction	(High-order word = Flags, Low-order word = NGreQueryEngineVersion)

Return Codes

This function returns the engine version number (lVersion) or, if an error occurs, GPI_ALTERROR.

Device Support Functions

- GreCreateBitmap
- GreDeleteBitmap
- GreSelectBitmap
- GreGetBitmapParameters
- GreGetBitmapDimension
- GreSetBitmapDimension
- GreGetAttributes
- GreSetAttributes
- GreSetGlobalAttribute
- GreGetDefaultAttributes
- GreSetDefaultAttributes
- GreGetDefaultArcParameters
- GreSetDefaultArcParameters
- GreGetDefaultViewingLimits
- GreSetDefaultViewingLimits
- GreInitializeAttributes.

```
HBITMAP GreCreateBitmap (hdc, pInfoHd, flUsage, pBitmap, paInfo, pInstance, lFunction)
```

Creates a bit map and returns its handle.

Support

This function is supported by the graphics engine. It is available for use by the presentation driver at InnerGreEntry.

Stack Frame

(HDC)	hdc	(device context handle)
(PBITMAPINFOHEADER)	pInfoHd	(pointer to BITMAPINFOHEADER structure for new bit map, see below)
(ULONG)	flUsage	(additional information, used when creating a new bit map, see below)
(PBYTE)	pBitmap	(pointer to bit-map initialization data)
(PBITMAPINFO)	paInfo	(pointer to BITMAPINFO structure for initialization data, see below)
(ULONG)	pInstance	(pointer to instance data)
(ULONG)	lFunction	(High-order word = flags, Low-order word = NGreCreateBitmap)

pInfoHd Pointer to an BITMAPINFOHEADER structure:

- **cbFix** Length, in bytes, of this structure.
- **cx** Width of bit map.
- **cy** Height of bit map.
- **cPlanes** Number of color planes.
- **cBitCount** Number of adjacent color bits per pel.

Each plane has ((cx*cBitCount+31)/32*4*cy) bytes.

flUsage The only defined flag is:

- **CBM_INIT** When set, the pBitmap and paInfo parameters are used to initialize the newly created bit map. If this flag is not set, the bit-map initialization is device dependent.

 When this flag is set, it is assumed that sufficient data is supplied to initialize the whole bit map.

 Other flags (16 through 31) can be used for special purposes known to be supported by a particular presentation driver.

paInfo BITMAPINFO structure:

- **cbFix** Length of structure.
- **cx** Bit-map width.
- **cy** Bit-map height.
- **cPlanes** Number of planes (1 if standard format).
- **cBitCount** Number of bits per pel.
- **argbColors[]** Color-table array of RGB structures:
 - **bBlue**
 - **bGreen**
 - **bRed.**

Note: See "Bit-Map File Formats" on page 12-30 for descriptions of the BITMAPINFO, BITMAPINFOHEADER, and bit-map data formats.

Return Codes

On completion, the handling routine must return the handle of the bit map (hbm) or zero if an error occurred. Error codes posted by the engine for this function include:

PMERR_HDC_BUSY
PMERR_DEV_FUNC_NOT_INSTALLED
PMERR_INSUFFICIENT_MEMORY
PMERR_INV_LENGTH_OR_COUNT
PMERR_INV_HDC
PMERR_INV_USAGE_PARM
PMERR_INV_INFO_TABLE
PMERR_INV_BITMAP_DIMENSION
PMERR_INV_SCAN_START.

Remarks

Bit-map size is limited by available memory, the maximum width and height are 64KB.

There are several standard bit-map formats that should normally be adhered to. These are:

Bitcount	Planes
1	1
4	1
8	1
24	1

Presentation drivers for display devices must support at least 4 bits per pel. For other devices, the driver must be able to create and accept any of the standard formats even though they might not use the color information.

The DC handle supplied to this function must never be null, bit maps always belong to some device. The bit map is created on the device specified, and can be selected to a different device later as the engine can handle the transfer of bits from one device to another.

When the value specified for cPlanes or cBitcount is incompatible with the physical device specified by the DC handle, an error is raised.

```
(BOOL) GreDeleteBitmap (hbm, pInstance, lFunction)
```

This function deletes the bit map identified by hbm. If the bit map is currently selected into a DC, GreDeleteBitmap returns an error.

Support

GreDeleteBitmap is supported by the graphics engine. It is available for use by the presentation driver at InnerGreEntry.

Stack Frame

(HBITMAP)	hbm	(bit-map handle)
(ULONG)	pInstance	(pointer to instance data)
(ULONG)	lFunction	(High-order word = flags, Low-order word = NGreDeleteBitmap)

Return Codes

This function returns BOOLEAN (fSuccess):

TRUE Successful.
FALSE Error.

When an error is detected, the graphics engine calls WinSetErrorInfo to post the condition. Reasons for failure of this function include:

PMERR_DEV_FUNC_NOT_INSTALLED
PMERR_INSUFFICIENT_MEMORY
PMERR_INV_LENGTH_OR_COUNT
PMERR_INV_HDC
PMERR_INV_HBITMAP
PMERR_BITMAP_IS_SELECTED
PMERR_HBITMAP_BUSY
PMERR_INV_INFO_TABLE
PMERR_INV_SCAN_START.

GreSelectBitmap

```
(HBITMAP) GreSelectBitmap (hdc, hbm, pInstance, lFunction)
```

GreSelectBitmap selects a bit map into a memory DC or, if called with a null bit-map handle, deselects the existing bit map from the DC. If this function is called to deselect a bit map, the return code is the handle of the deselected bit map.

Once created, a bit map has to be selected into a DC before the presentation driver can write to it. A bit map can be selected into the DC that created it or into any DC that has a compatible bit-map format. Compatibility can be ensured by using one of the standard formats.

The presentation driver must select a bit map into a device context before attempting to draw it.

Support

GreSelectBitmap is supported by the graphics engine. It is available for use by the presentation driver at InnerGreEntry.

Stack Frame

(HDC)	hdc	(device context handle)
(HBITMAP)	hbm	(bit-map handle)
(ULONG)	pInstance	(pointer to instance data)
(ULONG)	lFunction	(High-order word = flags, Low-order word = NGreSelectBitmap)

Return Codes

This function returns an HBITMAP value:

HBM_ERROR	Error
Null	Successful
Other	Handle of deselected bit-map.

An error is raised when:

- The bit map is incompatible with the DC and cannot be converted
- The bit map is already selected into a DC
- The bit map has been assigned a set ID for use as a pattern in an area fill operation.

Error codes posted by the engine for this function include:

PMERR_INV_IN_AREA
PMERR_INV_IN_PATH
PMERR_HDC_BUSY
PMERR_DEV_FUNC_NOT_INSTALLED
PMERR_INSUFFICIENT_MEMORY
PMERR_INV_LENGTH_OR_COUNT
PMERR_INV_RECT
PMERR_INV_COORDINATE
PMERR_INV_HDC
PMERR_INV_HRGN
PMERR_INV_REGION_CONTROL
PMERR_INV_HBITMAP
PMERR_BITMAP_IS_SELECTED
PMERR_HBITMAP_BUSY
PMERR_INV_INFO_TABLE
PMERR_INV_BITMAP_DIMENSION
PMERR_INV_SCAN_START.

```
(BOOL) GreGetBitmapParameters (hbm, pInfoHd, pInstance, lFunction)
```

This function returns, in the buffer addressed by pInfoHd, header information for the specified bit map. The header information is returned as a BITMAPINFOHEADER structure and gives details such as the width, height, number of planes, and number of bits per pel.

Support

This function is supported by the graphics engine. It is available for use by the presentation driver at InnerGreEntry.

Stack Frame

(HBITMAP)	hbm	(bit-map handle)
(PBITMAPINFOHEADER)	pInfoHd	(long pointer to BITMAPINFOHEADER structure, where the returned information is stored)
(ULONG)	pInstance	(pointer to instance data)
(ULONG)	lFunction	(High-order word = Flags, Low-order word = NGreGetBitmapParameters)

Return Codes

GreGetBitmapParameters returns BOOLEAN (fSuccess):

TRUE Successful.
FALSE Error.

When an error is detected, the graphics engine calls WinSetErrorInfo to post the condition. Reasons for failure of this function include:

PMERR_INV_HBITMAP
PMERR_BITMAP_IS_SELECTED
PMERR_HBITMAP_BUSY.

GreGetBitmapDimension

```
(BOOL) GreGetBitmapDimension (hbm, pDimension, pInstance, lFunction)
```

Renders height and width values for the bit map indicated by hbm. These are values which have been set by a previous GreSetBitmapDimension call. They are not used by the system.

Support

This function is supported by the graphics engine. It is available for use by the presentation driver at InnerGreEntry.

Stack Frame

(HBITMAP)	hbm	(bit-map handle)
(PSIZEL)	pDimension	(long pointer to width and height values, in 0.1 mm units)
(ULONG)	pInstance	(pointer to instance data)
(ULONG)	lFunction	(High-order word = Flags, Low-order word = NGreGetBitmapDimension)

Return Codes

This function returns BOOLEAN (fSuccess):

TRUE Successful.
FALSE Error.

When an error is detected, the graphics engine calls WinSetErrorInfo to post the condition. Reasons for failure of this function include:

PMERR_INV_HBITMAP
PMERR_BITMAP_IS_SELECTED
PMERR_HBITMAP_BUSY.

```
(BOOL) GreSetBitmapDimension (hbm, pDimension, pInstance, lFunction)
```

Associates height and width values for the bit map indicated by hbm. These values can be read back later by calling GreGetBitmapDimension.

Support

This function is supported by the graphics engine. It is available for use by the presentation driver at InnerGreEntry.

Stack Frame

(HBITMAP)	hbm	(bit-map handle)
(PSIZEL)	pDimension	(long pointer to width and height values, in 0.1 mm units, see below)
(ULONG)	pInstance	(pointer to instance data)
(ULONG)	lFunction	(High-order word = Flags, Low-order word = NGreSetBitmapDimension)

pDimension Pointer to a pair of parameters:

ulWidth Width of bit map, in 0.1 mm units.
ulHeight Height of bit map, in 0.1 mm units.

Return Codes

This function returns BOOLEAN (fSuccess):

TRUE Successful.
FALSE Error.

When an error is detected, the graphics engine calls WinSetErrorInfo to post the condition. Reasons for failure of this function include:

PMERR_INV_HBITMAP
PMERR_BITMAP_IS_SELECTED
PMERR_HBITMAP_BUSY.

GreGetAttributes

```
(LONG) GreGetAttributes (hdc, lPrimType, flAttrsMask, pAttrs, pInstance, lFunction)
```

Returns the current value of the attributes indicated in flAttrsMask. When a specified attribute is currently set to its default value, the corresponding flag in the returned defaults mask is set and the returned value for this attribute is undefined.

The engine either:

- Returns the value of each specified attribute, and resets the corresponding bit in the returned mask (flDefsMask) or,
- Sets the bit in the returned mask to indicate that the specified attribute is set to its default. The corresponding value in the attribute buffer is *not* valid and is likely to have been overwritten by the engine.

Support

This function is supported by the graphics engine. It is available for use by the presentation driver at InnerGreEntry.

Stack Frame

(HDC)	hdc	(device-context handle)
(LONG)	lPrimType	(bundle primitive type, see below)
(ULONG)	flAttrsMask	(attribute mask, see below)
(PBUNDLE)	pAttrs	(pointer to the fixed format bundle record containing the attributes returned)
(ULONG)	pInstance	(pointer to instance data)
(ULONG)	lFunction	(High-order word = Flags, Low-order word = NGreGetAttributes)

lPrimType Indicates the bundle type. Valid primitive values are:

PRIM_LINE Line attribute bundle.
PRIM_CHAR Character attribute bundle.
PRIM_MARKER Marker attribute bundle.
PRIM_AREA Pattern attribute bundle.
PRIM_IMAGE Image attribute bundle.

flAttrsMask Specifies the attributes to be returned. This mask contains a bit corresponding to each attribute in the bundle record. For each set bit, the engine returns the corresponding attribute values and default mask bits.

pAttrs The returned attribute value (bundle). The only fields that are updated are those whose corresponding flags in flAttrsMask have been set.

Return Codes

This function returns the default attribute bit mask (flDefsMask):

flDefsMask Default bit mask. Only those bits with corresponding set bits in flAttrsMask are updated.
GPI_ALTERROR Error.

When an error is detected, the graphics engine calls WinSetErrorInfo to post the condition. Reasons for failure of this function include:

PMERR_HDC_BUSY
PMERR_INV_HDC.

```
(BOOL) GreSetAttributes (hdc, lPrimType, flDefsMask, flAttrsMask, pAttrs, pInstance, lFunction)
```

Sets the attributes for the specified primitive type according to the flags set in the flDefsMask and flAttrsMask parameters:

- Only those attributes whose flags are set in flAttrsMask are modified.
- Those attributes whose flags are set in both flAttrsMask and flDefsMask are set to their default values.
- Those attributes whose flags are set in flDefsMask only, or whose flags are not set either in flDefsMask or flAttrsMask, are unchanged.
- Those attributes whose flags are set in flAttrsMask only are set to the value specified by pAttrs.

When this function occurs within a path bracket, it must not be used to set the geometric line width for line attributes, nor must it be used to set the foreground and background colors and mixes for character and marker attributes.

Support

This function is supported by the graphics engine. It is available for use by the presentation driver at InnerGreEntry.

Stack Frame

(HDC)	hdc	(device context handle)
(LONG)	lPrimType	(bundle primitive type, see below)
(ULONG)	flDefsMask	(flags indicating which attributes are to be set to default)
(ULONG)	flAttrsMask	(flags indicating which attributes are to be modified)
(PBUNDLE)	pAttrs	(pointer to the fixed-format bundle record containing the attribute values to be set, see below)
(ULONG)	pInstance	(pointer to instance data)
(ULONG)	lFunction	(High-order word = Flags, Low-order word = NGreSetAttributes)

lPrimType Indicates the bundle type. Valid primitive values are:

PRIM_LINE	Pen (line attribute bundle).
PRIM_CHAR	Character attribute bundle.
PRIM_MARKER	Marker attribute bundle.
PRIM_AREA	Pattern attribute bundle.
PRIM_IMAGE	Image attribute bundle.

pAttrs This is a long pointer to the fixed format-bundle record containing the attribute values to be set, as specified by flAttrsMask. Only the attribute fields corresponding to attribute flags set in flAttrsMask and not set in flDefsMask, contain valid values. This buffer need only be large enough to contain data for the highest offset attribute referenced.

Return Codes

This function returns BOOLEAN (fSuccess):

TRUE	Successful.
FALSE	Error.

When an error is detected, the graphics engine calls WinSetErrorInfo to post the condition. Reasons for failure of this function include:

PMERR_INV_IN_AREA
PMERR_HDC_BUSY
PMERR_DEV_FUNC_NOT_INSTALLED
PMERR_COORDINATE_OVERFLOW
PMERR_INV_PRIMITIVE_TYPE
PMERR_INV_SETID
PMERR_INV_CHAR_SET_ATTR

PMERR_INV_MARKER_SET_ATTR
PMERR_INV_PATTERN_SET_ATTR
PMERR_INV_MARKER_SYMBOL_ATTR
PMERR_UNSUPPORTED_ATTR
PMERR_INV_MIX_ATTR
PMERR_INV_BACKGROUND_MIX_ATTR
PMERR_INV_PATTERN_REF_PT_ATTR
PMERR_INV_LINE_WIDTH_ATTR
PMERR_INV_GEOM_LINE_WIDTH_ATTR
PMERR_INV_LINE_TYPE_ATTR
PMERR_INV_LINE_JOIN_ATTR
PMERR_INV_LINE_END_ATTR
PMERR_INV_CHAR_MODE_ATTR
PMERR_INV_CHAR_SHEAR_ATTR
PMERR_INV_CHAR_DIRECTION_ATTR
PMERR_UNSUPPORTED_ATTR_VALUE
PMERR_INSUFFICIENT_MEMORY
PMERR_INV_LENGTH_OR_COUNT
PMERR_INV_CODEPAGE
PMERR_EXCEEDS_MAX_SEG_LENGTH
PMERR_INV_HDC
PMERR_INV_COORD_SPACE
PMERR_INV_PATTERN_SET_FONT
PMERR_HUGE_FONTS_NOT_SUPPORTED
PMERR_INV_COLOR_ATTR
PMERR_INV_BACKGROUND_COL_ATTR.

When this function is called within an area definition, an error is raised by the engine and the condition PMERR_INV_IN_AREA is posted.

```
(BOOL) GreSetGlobalAttribute (hdc, lAttrType, lAttribute, flOptions, pInstance, lFunction)
```

This function sets the specified attribute in the pen, pattern, character, image, and marker bundles. The attribute can be set to its default value or to a specified value.

Support

GreSetGlobalAttribute is supported by the graphics engine. It is available for use by the presentation driver at InnerGreEntry.

Stack Frame

(HDC)	hdc	(device context handle)
(LONG)	lAttrType	(specifies the attribute, see below)
(LONG)	lAttribute	(new attribute value)
(ULONG)	flOptions	(see below)
(ULONG)	pInstance	(pointer to instance data)
(ULONG)	lFunction	(High-order word = Flags, Low-order word = NGreSetGlobalAttribute)

lAttrType Attribute type:

ATYPE_COLOR Foreground color.
ATYPE_BACK_COLOR Background color.
ATYPE_MIX_MODE Foreground mix.
ATYPE_BACK_MIX_MODE Background mix.

Note: ATYPE_BACK_COLOR and ATYPE_BACK_MIX_MODE do not apply to the line bundle.

flOptions The only allowable option flag is GATTR_DEFAULT; this specifies that the attribute indicated by lAttrType should be set to its default value. If the GATTR_DEFAULT flag is not set, the function sets the attribute to the value specified by lAttribute.

Return Codes

On completion, the handling routine returns BOOLEAN (fSuccess):

TRUE Successful.
FALSE Error.

When an error is detected, the graphics engine calls WinSetErrorInfo to post the condition. Reasons for failure of this function include:

PMERR_INV_IN_AREA
PMERR_HDC_BUSY
PMERR_DEV_FUNC_NOT_INSTALLED
PMERR_INV_RESET_OPTIONS
PMERR_INV_MIX_ATTR
PMERR_INV_BACKGROUND_MIX_ATTR
PMERR_INV_HDC
PMERR_INV_COLOR_ATTR
PMERR_INV_BACKGROUND_COL_ATTR.

GreGetDefaultAttributes

```
(BOOL) GreGetDefaultAttributes (hdc, lPrimType, flDefsMask, flAttrsMask, pAttrs, pInstance, lFunction)
```

Returns the default values of the attributes indicated in flAttrsMask.

Support

This function is supported by the graphics engine. It is available for use by the presentation driver at InnerGreEntry.

Stack Frame

(HDC)	hdc	(device context handle)
(LONG)	lPrimType	(bundle primitive type, see below)
(ULONG)	flAttrsMask	(attribute mask, see below)
(PBUNDLE)	pAttrs	(pointer to the fixed format bundle record containing the attributes returned)
(ULONG)	pInstance	(pointer to instance data)
(ULONG)	lFunction	(High-order word = Flags, Low-order word = NGreGetDefaultAttributes)

lPrimType Indicates the bundle type. Valid primitive values are:

PRIM_LINE	Pen (line attribute bundle).
PRIM_CHAR	Character attribute bundle.
PRIM_MARKER	Marker attribute bundle.
PRIM_AREA	Pattern attribute bundle.
PRIM_IMAGE	Image attribute bundle.

flAttrsMask Specifies the attributes to be returned. This mask contains a bit corresponding to each attribute in the bundle record. For each set bit, the engine returns the corresponding attribute values and default mask bits.

Return Codes

This function returns the default attribute bit mask (flDefsMask):

flDefsMask Default bit mask. Only those bits with corresponding set bits in flAttrsMask are updated.

ATTRS_ERROR Error.

When an error is detected, the graphics engine calls WinSetErrorInfo to post the condition. Reasons for failure of this function include:

PMERR_HDC_BUSY
PMERR_INV_HDC.

```
(BOOL) GreGetDefaultAttributes (hdc, lPrimType, flDefsMask, flAttrsMask, pAttrs, pInstance, lFunction)
```

Sets the attributes for the specified primitive type according to the flags set in the flDefsMask and flAttrsMask parameters to the default values.

- Only those attributes whose flags are set in flAttrsMask are modified.
- Those attributes whose flags are set in both flAttrsMask and flDefsMask are set to their default values.
- Those attributes whose flags are set in flDefsMask only, or whose flags are not set either in flDefsMask or flAttrsMask, are unchanged.
- Those attributes whose flags are set in flAttrsMask only are set to the value specified by pAttrs.

When this function occurs within a path bracket, it must not be used to set the geometric line width for line attributes, nor must it be used to set the foreground and background colors and mixes for character and marker attributes.

Support

This function is supported by the graphics engine. It is available for use by the presentation driver at InnerGreEntry.

Stack Frame

(HDC)	hdc	(device context handle)
(LONG)	lPrimType	(bundle primitive type, see below)
(ULONG)	flDefsMask	(flags indicating which attributes are to be set to default)
(ULONG)	flAttrsMask	(flags indicating which attributes are to be modified)
(PBUNDLE)	pAttrs	(pointer to the fixed-format bundle record containing the attribute values to be set, see below)
(ULONG)	pInstance	(pointer to instance data)
(ULONG)	lFunction	(High-order word = Flags, Low-order word = NGreSetDefaultAttributes)

lPrimType Indicates the bundle type. Valid primitive values are:

PRIM_LINE	Line attribute bundle.
PRIM_CHAR	Character attribute bundle.
PRIM_MARKER	Marker attribute bundle.
PRIM_AREA	Pattern attribute bundle.
PRIM_IMAGE	Image attribute bundle.

pAttrs This is a long pointer to the fixed format-bundle record containing the attribute values to be set, as specified by flAttrsMask. Only the attribute fields corresponding to attribute flags set in flAttrsMask and not set in flDefsMask, contain valid values.

Return Codes

This function returns BOOLEAN (fSuccess):

TRUE	Successful.
FALSE	Error.

When an error is detected, the graphics engine calls WinSetErrorInfo to post the condition. Reasons for failure of this function include:

PMERR_HDC_BUSY
PMERR_DEV_FUNC_NOT_INSTALLED
PMERR_COORDINATE_OVERFLOW
PMERR_INV_PRIMITIVE_TYPE
PMERR_INV_SETID
PMERR_INV_CHAR_SET_ATTR
PMERR_INV_MARKER_SET_ATTR
PMERR_INV_PATTERN_SET_ATTR

PMERR_INV_MARKER_SYMBOL_ATTR
PMERR_UNSUPPORTED_ATTR
PMERR_INV_MIX_ATTR
PMERR_INV_BACKGROUND_MIX_ATTR
PMERR_INV_PATTERN_REF_PT_ATTR
PMERR_INV_LINE_WIDTH_ATTR
PMERR_INV_GEOM_LINE_WIDTH_ATTR
PMERR_INV_LINE_TYPE_ATTR
PMERR_INV_LINE_JOIN_ATTR
PMERR_INV_LINE_END_ATTR
PMERR_INV_CHAR_MODE_ATTR
PMERR_INV_CHAR_SHEAR_ATTR
PMERR_INV_CHAR_DIRECTION_ATTR
PMERR_UNSUPPORTED_ATTR_VALUE
PMERR_INSUFFICIENT_MEMORY
PMERR_INV_LENGTH_OR_COUNT
PMERR_INV_CODEPAGE
PMERR_EXCEEDS_MAX_SEG_LENGTH
PMERR_INV_HDC
PMERR_INV_COORD_SPACE
PMERR_INV_PATTERN_SET_FONT
PMERR_HUGE_FONTS_NOT_SUPPORTED
PMERR_INV_COLOR_ATTR
PMERR_INV_BACKGROUND_COL_ATTR.

```
(BOOL) GreGetArcParameters (hdc, pArcParms, pInstance, lFunction)
```

Stores the default arc parameters in the buffer addressed by pArcParms.

Support

This function is supported by the graphics engine and can be hooked by the presentation driver.

Stack Frame

(HDC)	hdc	(device-context handle)
(PARCPARAMS)	pArcParms	(pointer to ARCPARAMS structure)
(ULONG)	pInstance	(pointer to instance data)
(ULONG)	lFunction	(High-order word = Flags, Low-order word = NGreGetDefaultArcParameters)

Return Codes

On completion, the handling routine must return BOOLEAN (fSuccess):

TRUE Successful.
FALSE Error.

When an error is detected, the handling routine must call WinSetErrorInfo to post the condition. Error codes for conditions that the handling routine is expected to check include:

PMERR_HDC_BUSY
PMERR_INV_HDC.

GreSetDefaultArcParameters

```
(BOOL) GreSetArcParameters (hdc, pArcParms, pInstance, lFunction)
```

Sets the arc parameters to the default values.

Support

This function is supported by the graphics engine and can be hooked by the presentation driver.

Stack Frame

(HDC)	hdc	(device-context handle)
(PULONG)	(PARCPARAMS)	(pointer to the default arc parameters, see below)
(ULONG)	pInstance	(pointer to instance data)
(ULONG)	lFunction	(High-order word = Flags, Low-order word = NGreSetDefaultArcParameters)

pArcParms This is an ARCPARAMS structure containing the default arc parameters. See the *Programming Reference* for a full description of the arc parameters structure.

Return Codes

On completion, the handling routine must return BOOLEAN (fSuccess):

TRUE Successful.
FALSE Error.

When an error is detected, the handling routine must call WinSetErrorInfo to post the condition. Error codes for conditions that the handling routine is expected to check include:

PMERR_HDC_BUSY
PMERR_INV_HDC.

```
(BOOL) GreGetViewingLimits (hdc, prcViewingLimits, pInstance, lFunction)
```

Loads the array, indicated by prcViewingLimits, with the default boundaries of the viewing window in graphic model space coordinates.

Support

This function is supported by the graphics engine and can be hooked by the presentation driver.

Stack Frame

(HDC)	hdc	(device-context handle)
(PRECTL)	prcViewLimits	(pointer to limits of viewing area, see below)
(ULONG)	pInstance	(pointer to instance data)
(ULONG)	lFunction	(High-order word = Flags, Low-order word = NGreGetDefaultViewingLimits)

prcViewLimits RECTL structure:

xLeft Minimum x coordinate of viewing limits
yBottom Minimum y coordinate
xRight Maximum x coordinate of viewing limits
yTop Maximum y coordinate.

Return Codes

On completion, the handling routine must return BOOLEAN (fSuccess):

TRUE Successful.
FALSE Error.

When an error is detected, the handling routine must call WinSetErrorInfo to post the condition. Error codes for conditions that the handling routine is expected to check include:

PMERR_HDC_BUSY
PMERR_INV_HDC.

GreSetDefaultViewingLimits

```
(BOOL) GreSetViewingLimits (hdc, prcViewingLimits, pInstance, lFunction)
```

Sets the boundaries of the default viewing (clip) limits to the values specified by prcViewingLimits. The current viewing limits (see "GreGetViewingLimits" on page 14-97) are unaffected by this call.

Support

This function is supported by the graphics engine.

Stack Frame

(HDC)	hdc	(device-context handle)
(PRECTL)	prcViewLimits	(pointer to limits of viewing area, see below)
(ULONG)	pInstance	(pointer to instance data)
(ULONG)	lFunction	(High-order word = Flags, Low-order word = NGreSetDefaultViewingLimits)

prcViewLimits RECTL structure:

xLeft Minimum x coordinate of viewing limits
yBottom Minimum y coordinate
xRight Maximum x coordinate of viewing limits
yTop Maximum y coordinate.

Return Codes

On return, the handling routine must return BOOLEAN (fSuccess):

TRUE Successful.
FALSE Error.

When an error is detected, the handling routine must call WinSetErrorInfo to post the condition. Error codes for conditions that the handling routine is expected to check include:

PMERR_INV_IN_AREA
PMERR_INV_IN_PATH
PMERR_HDC_BUSY
PMERR_INV_COORDINATE
PMERR_INV_HDC
PMERR_INV_GRAPHICS_FIELD.

```
(BOOL) GreInitializeAttributes (hdc, flOptions, pInstance, lFunction)
```

Sets the current default attributes to the initial standard default values. It can also be used to reset the current attributes to the current default values.

This function affects all bundle attributes that can be set with GreSetAttributes, arc parameters, and viewing limits.

Support

This function is supported by the graphics engine.

Stack Frame

(HDC)	hdc	(device-context handle)
(ULONG)	flOptions	(option flags, see below)
(ULONG)	pInstance	(pointer to instance data)
(ULONG)	lFunction	(High-order word = Flags, Low-order word = NGreInitializeAttributes)

flOptions Valid flags are:

INAT_DEFAULTATTRIBUTES When set, all the current default attributes are set to their initial standard default values.

INAT_CURRENTATTRIBUTES When set, all the current attributes are set to their current default values.

When both flags are set, INAT_DEFAULTATTRIBUTES is processed first.

Return Codes

On completion, the handling routine must return BOOLEAN (fSuccess):

TRUE Successful.
FALSE Error.

Set ID Functions

- GreDeleteSetId
- GreQueryNumberSetIds
- GreQuerySetIds.

```
(BOOL) GreDeleteSetId (hdc, lcid, pInstance, lFunction)
```

This function deletes the character set, marker set, or pattern set identified by lcid.

Base sets cannot be deleted. An error is logged if GreDeleteSetId is called to delete the current character, marker, or pattern set.

Support

GreDeleteSetId is supported by the graphics engine. It is available for use by the presentation driver at InnerGreEntry.

Stack Frame

(HDC)	hdc	(device-context handle)
(LONG)	lcid	(local identifier, see below)
(ULONG)	pInstance	(pointer to instance data)
(ULONG)	lFunction	(High-order word = Flags, Low-order word = NGreDeleteSetId)

lcid When this identifies a bit map, only the lcid is deleted; the the bit map will have no lcid but it will still exist.

When lcid = LCID_ALL, all loaded graphics local identifiers, such as logical fonts and bit map IDs, are destroyed. In this instance AVIO fonts are unaffected, these can only be deleted explicitly.

LCID_AVIO_1, LCID_AVIO_2, and LCID_AVIO_3 represent AVIO fonts 1, 2, and 3 respectively.

Return Codes

This function returns BOOLEAN (fSuccess):

TRUE Successful.
FALSE Error.

When an error is detected, the graphics engine calls WinSetErrorInfo to post the condition. Reasons for failure of this function include:

PMERR_INV_IN_AREA
PMERR_HDC_BUSY
PMERR_DEV_FUNC_NOT_INSTALLED
PMERR_COORDINATE_OVERFLOW
PMERR_INV_SETID
PMERR_INSUFFICIENT_MEMORY
PMERR_INV_LENGTH_OR_COUNT
PMERR_INV_CODEPAGE
PMERR_SETID_IN_USE
PMERR_INV_HDC
PMERR_INV_COORD_SPACE
PMERR_SETID_NOT_FOUND
PMERR_INV_FONTDEF
PMERR_INV_EXTENDED_LCID.

GreQueryNumberSetIds

```
(LONG) GreQueryNumberSetIds (hdc, lRange, pInstance, lFunction)
```

Returns the total number of LCIDs, such as logical fonts and bit-map IDs, that have been created.

Support

This function is supported by the graphics engine. It is available for use by the presentation driver at InnerGreEntry.

Stack Frame

(HDC)	hdc	(device context handle)
(ULONG)	lRange	(indicates whether GPI, or AVIO LCIDs, or both, are to be returned, see below)
(ULONG)	pInstance	(pointer to instance data)
(ULONG)	lFunction	(High-order word = Flags, Low-order word = NGreQueryNumberSetIds)

lRange Valid ranges are:

LCID_RANGE_GPI	GPI
LCID_RANGE_AVIO	AVIO
LCID_RANGE_BOTH	GPI and AVIO.

Return Codes

This function returns the number of LCIDs (cLcid) or GPI_ALTERROR if an error occurred.

When an error is detected, the graphics engine calls WinSetErrorInfo to post the condition. Reasons for failure of this function include:

PMERR_HDC_BUSY
PMERR_INV_SETID
PMERR_INV_HDC
PMERR_INV_SETID_TYPE.

```
(BOOL) GreQuerySetIds (hdc, cSets, paTypes, paszNames, paLcid, lRange, pInstance, lFunction)
```

This function returns a list of created LCIDs with their names and types in the buffers addressed by paLcid, paszNames, and paTypes.

Support

GreQuerySetIds is supported by the graphics engine. It is available for use by the presentation driver at InnerGreEntry.

Stack Frame

(HDC)	hdc	(device context handle)
(LONG)	cSets	(number of sets to return)
(PLONG)	paTypes	(pointer to an array of cSets elements indicating the type of the corresponding set, see below)
(PSTR8)	paszNames	(pointer to an array of 8-character names associated with the corresponding LCIDs, for a bit map the name is filled with zeroes)
(PLONG)	paLcid	(pointer to an array of cSets elements to which the lcids are returned)
(ULONG)	lRange	(indicates whether GPI, or AVIO local identifiers, or both, are to be returned, see below)
(ULONG)	pInstance	(pointer to instance data)
(ULONG)	lFunction	(High-order word = Flags, Low-order word = NGreQuerySetIds)

paTypes Each element in this array is a LONG value indicating the type of the corresponding local identifier:

LCIDT_FONT	Logical font
LCIDT_BITMAP	Bit-map ID.

lRange Valid ranges are:

LCID_RANGE_GPI	GPI
LCID_RANGE_AVIO	AVIO
LCID_RANGE_BOTH	GPI and AVIO.

LCID_AVIO_1, LCID_AVIO_2, and LCID_AVIO_3 represent AVIO sets of 1, 2, and 3.

Return Codes

This function returns BOOLEAN (fSuccess):

TRUE	Successful.
FALSE	Error.

When an error is detected, the graphics engine calls WinSetErrorInfo to post the condition. Reasons for failure of this function include:

PMERR_HDC_BUSY
PMERR_INV_SETID
PMERR_INV_LENGTH_OR_COUNT
PMERR_INV_HDC
PMERR_INV_SETID_TYPE.

Remarks

When cSets is greater than the number of lcids, the unused elements of paTypes and paLcid are set to zero.

LCID Functions

- GreQueryBitmapHandle
- GreSetBitmapID.

```
(HBITMAP) GreQueryBitmapHandle (hdc, lLcid, pInstance, lFunction)
```

Gets the bit-map handle for the specified local identifier (lcid). When lLcid does not reference a bit map, an error is raised by the engine.

Support

This function is supported by the graphics engine. It is available for use by the presentation driver at InnerGreEntry.

Stack Frame

(HDC)	hdc	(device-context handle)
(LONG)	lLcid	(local identifier for which the bit-map handle is required)
(ULONG)	pInstance	(pointer to instance data)
(ULONG)	lFunction	(High-order word = Flags, Low-order word = NGreQueryBitmapHandle)

Return Codes

This function returns the bit-map handle (hbm) or, if an error occurs, GPI_ERROR.

When an error is detected, the graphics engine calls WinSetErrorInfo to post the condition. Reasons for failure of this function include:

PMERR_HDC_BUSY
PMERR_BITMAP_NOT_SELECTED
PMERR_INV_SETID
PMERR_INV_HDC.

GreSetBitmapID

```
(BOOL) GreSetBitmapID (hdc, hbm, lLcid, pInstance, lFunction)
```

This function sets the local identifier (lcid) for a bit map. The presentation driver needs to assign an lcid before the bit map can be used for area shading or as the pattern in a BitBlt operation. The bit map can be of any format supported by the device; however, it may be simplified by the engine before use.

Errors are raised by the engine when:

- The local identifier is already in use
- The bit map is already selected into a memory DC.

Note: When a bit map is destroyed, its lcid becomes undefined.

Support

GreSetBitmapID is supported by the graphics engine. It is available for use by the presentation driver at InnerGreEntry.

Stack Frame

(HDC)	hdc	(device context handle)
(HBITMAP)	hbm	(bit-map handle)
(LONG)	lLcid	(local identifier to be associated with the bit map)
(ULONG)	pInstance	(pointer to instance data)
(ULONG)	lFunction	(High-order word = Flags, Low-order word = NGreSetBitmapID)

Return Codes

This function returns BOOLEAN (fSuccess):

TRUE Successful.
FALSE Error.

When an error is detected, the graphics engine calls WinSetErrorInfo to post the condition. Reasons for failure of this function include:

PMERR_HDC_BUSY
PMERR_INV_HDC
PMERR_INV_HBITMAP
PMERR_BITMAP_IS_SELECTED
PMERR_HBITMAP_BUSY.

```
(BOOL) GreCopyDCLoadData (hdc, flCmd, hdcSrc, pInstance, lFunction)
```

Copies the loaded fonts, bit maps, color table, and default attributes from one DC to another. Device-dependant attributes, such as pick aperture, character cell, and marker cell are only copied when they have been set by the application, otherwise the target device defaults are used.

In response to this call, the graphics engine does the following:

- Transfers the contents of the source DC's lcid table to the target DC's lcid table.
- Translates any bit maps, when the devices associated with the source and target DCs are different.
- Calls GreCreateLogicalFont for the target DC, as necessary, see below.
- Transfers the color table and checks that the resulting color table in the target DC is the same as the one in the source DC.
- Resets the source DC using GreResetDC. When the save level of the source DC is not 1, it is restored to the default value before GreResetDC is called. The color table is also reset to the default value.

The values of the GPI handles in the source DC are preserved. This function fails if the target DC lcid table already has lcids set in the range specified.

If the target DC does not support LCOL_REALIZABLE (see "GreCreateLogColorTable" on page 12-96), a warning (PMERR_REALIZE_NOT_SUPPORTED) is raised and the color table is treated as non-realizable.

The effect of this function on fonts is:

- Any logical font in force on the source DC is reloaded onto the target, with the original parameters specified by the application.
- A generic font, selected by match ID, can be selected into the target DC, provided that it is suitable. For example an image font cannot be select for a plotter DC.
- Any device font, selected by match ID, can be selected for the target DC, provided that the new device has a font with the same match ID. If not, the defaults are used.

 It is the responsibility of the presentation driver to manage this. When it is necessary to use a specific font on the new device, the presentation driver should call GreQueryFonts to determine the characteristics and match ID of the font for the new device and then to reload it.
- A warning is raised when it is not possible to reload a font on association (not on reassociation). This warning is one of:
 PMERR_FONT_NOT_LOADED
 PMERR_KERNING_NOT_SUPPORTED.

Support

This function is supported by the graphics engine.

Stack Frame

(HDC)	hdc	(handle of target DC)
(ULONG)	flCmd	(flags indicating which fonts to copy, see below)
(HDC)	hdcSrc	(handle of source DC)
(ULONG)	pInstance	(pointer to instance data)
(ULONG)	lFunction	(High-order word = Flags, Low-order word = NGreCopyDCLoadData)

flCmd Valid flags are:

LCID_RANGE_GPI	GPI
LCID_RANGE_AVIO	AVIO
LCID_RANGE_BOTH	GPI and AVIO.

GreCopyDCLoadData

Return Codes

This function returns BOOLEAN (fSuccess):

TRUE	Successful.
FALSE	Error.

Font Functions

- GreQueryLogicalFont
- GreCreateLogicalFont
- GreLoadFont
- GreUnLoadFont
- GreQueryFonts
- GreQueryFontAttributes
- GreQueryCodePageVector
- GreQueryFontFileDescriptions.

When Presentation Manager is first initialized, the engine is responsible for establishing the default image and outline fonts and the default pattern and marker sets. The presentation driver can supply device fonts as default fonts; if this is done, the driver can request the graphics engine to manage the device fonts by setting the CAPS_FONT_OUTLINE_MANAGE or CAPS_FONT_IMAGE_MANAGE flag in the array of device capabilities (see "GreQueryDeviceCaps" on page 12-106). When these flags are set, the engine performs any mapping and transforms that may be necessary. To use a default font belong to the driver, the font handle is passed to the engine in response to GreQueryDevResource (see page 12-108).

Font Matching

When requested by an application, the mapping of a font to a loaded font that is physically available, is done by the graphics engine. A physical font can be one that is built into a particular device, or a public font, loaded from a .FON file when Presentation Manager is first initialized.

The engine matches fonts by checking the values for szFacename and IMatch, in the font-metrics structures, with those passed to it in GreCreateLogicalFont. If no match is found, the engine proceeds as if the match number was passed with a value of zero. When a value of zero is passed for either szFacename or IMatch, the engine ignores that parameter and considers it to match anything. If no face name is specified, the engine supplies a default font from the presentation driver (see "GreQueryDevResource" on page 12-108) or, if none is available, maps the requested font to one of its own default fonts.

When a face name is specified, and the match number is zero, the engine checks to see if an image font was requested and, if true, then searches for a font of the specified dimensions. If this search fails, the engine then attempts to match the font to an outline font with the face name and selection flags requested. When the match number is zero, and the required font is an outline font, the engine searches for an outline font. Should all these searches fail, a default font is used. See "GreRealizeFont" on page 12-72.

GreQueryLogicalFont

```
(BOOL) GreQueryLogicalFont (hdc, lcid, pchName, pLogFont, cLogFont, pInstance, lFunction)
```

This function returns the 8-character name and attributes for the logical font that is defined for the specified lcid. The data is returned in the locations addressed by pchName and pLogFont.

When lcid identifies a bit-map ID, an error is raised by the engine.

Support

GreQueryLogicalFont is supported by the graphics engine. It is available for use by the presentation driver at InnerGreEntry.

Stack Frame

(HDC)	hdc	(device-context handle)
(LONG)	lcid	(local identifier for the logical font)
(PSTR8)	pchName	(pointer to an 8-character name used to describe the logical font)
(PFATTRS)	pLogFont	(pointer to a font-attribute structure)
(ULONG)	cLogFont	(number of bytes of font-attribute information requested)
(ULONG)	pInstance	(pointer to instance data)
(ULONG)	lFunction	(High-order word = Flags, Low-order word = NGreQueryLogicalFont)

Return Codes

This function returns BOOLEAN (fSuccess):

TRUE Successful.
FALSE Error.

When an error is detected, the graphics engine calls WinSetErrorInfo to post the condition. Reasons for failure of this function include:

PMERR_HDC_BUSY
PMERR_INV_SETID
PMERR_INV_LENGTH_OR_COUNT
PMERR_INV_HDC.

```
(LONG) GreCreateLogicalFont (hdc, lcid, pchName, pAttrs, pInstance, lFunc
```

This function sets the local identifier (lcid) for a logical font. The parameters passed to the function identify the name and attributes of the required font. The graphics engine selects a font, from the list of available fonts, that provides the best match for the font attributes addressed by pAttrs.

The selection is made in one of two ways:

1. If the lMatch attribute is non-zero, the calling program has already determined, from a call to GreQueryFonts, which font it requires. In this case the engine selects the font identified by the lMatch and szFaceName attributes or, if szFaceName is not specified, lMatch alone.
2. If lMatch is zero or a suitable match could not be found, the system examines the other fields in the attributes structure and selects the font that gives the best match.

If no match is found, the default font is used.

Once assigned, the system will not change the relationship of a specific lcid to a specific font.

Support

GreCreateLogicalFont is supported by the graphics engine. It is available for use by the presentation driver at InnerGreEntry.

Stack Frame

(HDC)	hdc	(device-context handle)
(LONG)	lcid	(local identifier that is to be assigned to the font)
(PSTR8)	pchName	(pointer to an eight-character name, used to describe the logical font)
(PFATTRS)	pAttrs	(pointer to font attributes structure, see below)
(ULONG)	pInstance	(pointer to instance data)
(ULONG)	lFunction	(High-order word = Flags, Low-order word = NGreCreateLogicalFont)

pAttrs Pointer to a FATTRS structure containing the font attributes:

usRecordLength Length in bytes, of this structure.

fsSelection Flags that can be set to define those features to be simulated by the system:

FATTR_SEL_ITALIC Italic characters are required.

FATTR_SEL_UNDERSCORE Underscored characters are required.

FATTR_SEL_STRIKEOUT Strikeout characters are required.

FATTR_SEL_BOLD Bold characters are required.

lMatch The match number for the font. GreQueryFonts returns a unique match number for each font which, together with szFaceName, can be used for font selection.

If the match number is negative, a device font is selected; if it is positive, an engine font is selected.

szFaceName The typeface name to which the font is designed. If the szFaceName is provided then an attempt is made to select the font with that face name. If the szFaceName is blank (0 length or NULL) then a default font is used.

idRegistry The Registry number for the font. This should be set to the value returned by GreQueryFonts.

usCodePage Defines the code page supported by the font.

lMaxBaselineExt The required (average) height above the baseline for uppercase characters, in world coordinates. For outline and transformable fonts (FATTR_FONTUSE_OUTLINE and FATTR_FONTUSE_TRANSFORM), lMaxBaselineExt should be set to zero.

lAveCharWidth For image fonts, the required average inter-character increment for the font in world coordinates. For outline and transformable fonts (FATTR_FONTUSE_OUTLINE and FATTR_FONTUSE_TRANSFORM), lAveCharWidth should be set to zero.

fsType Type indicator. Setting fsType to FATTR_TYPE_KERNING indicates that kerning is to be used if the font provides kerning information.

Other bits Reserved.

fsFontUse Flags containing information about how the font is to be related to the character attributes:

FATTR_FONTUSE_NOMIX Specifies that permissive mixing is allowed when the font is used. The engine can ignore any interaction with graphics primitives and can use overpaint and leave alone as the foreground and background mixes rather than using the current mix attributes.

FATTR_FONTUSE_OUTLINE Specifies that the font must be an outline font.

When the font is not defined by lMatch and FATTR_FONTUSE_OUTLINE is specified, the system searches for an outline font. If the search fails, a default font is selected. When the font is not defined by lMatch, and FATTR_FONTUSE_OUTLINE is not set, the system searches for an image font that matches lMaxBaselineExt and lAveCharWidth. If this fails it searches for an outline font with the required szFaceName and fsSelection flags.

FATTR_FONTUSE_TRANSFORMABLE Specifies that the font must be transformable (that is, it can be scaled, rotated, and sheared. Transformable fonts can only be used in CM_MODE3; non-transformable fonts can be used in CM_MODE1 or CM_MODE2 but not in CM_MODE3.

Return Codes

This function returns a LONG value indicating whether the font has been matched:

FONT_MATCH The font has been matched successfully.
FONT_DEFAULT The font was not matched, default font is to be used.
GPI_ERROR Error.

When an error is detected, the graphics engine calls WinSetErrorInfo to post the condition. Reasons for failure of this function include:

PMERR_BASE_ERROR
PMERR_INV_IN_AREA
PMERR_HDC_BUSY
PMERR_DEV_FUNC_NOT_INSTALLED
PMERR_COORDINATE_OVERFLOW
PMERR_INV_SETID
PMERR_INSUFFICIENT_MEMORY
PMERR_INV_LENGTH_OR_COUNT
PMERR_INV_CODEPAGE
PMERR_EXCEEDS_MAX_SEG_LENGTH
PMERR_SETID_IN_USE
PMERR_INV_HDC
PMERR_INV_COORD_SPACE
PMERR_INV_FONT_ATTRS
PMERR_INV_FONTDEF
PMERR_INV_EXTENDED_LCID.

Description

The interaction between fonts and character attributes depends on the state of the FATTRS_FONTUSE_TRANSFORMABLE flag in the font attributes structure. When this flag is set:

- The size of the characters is determined by the values of the character attributes at the time that the characters are drawn.
- The characters are positioned, rotated, and sheared as required.
- No checking is done.
- Any transformation is done by mapping the box defined by the font metrics parameters xMaxCharInc and yEmHeight to the character box, under the influence of character angle and shear.

When the FATTRS_FONTUSE_TRANSFORMABLE flag is not set, the lAveCharWidth and lBaselineExt parameters in the font attributes structure define the size of the font to be used; the character box attribute has no effect.

Transformable fonts cannot be used in character modes 1 and 2; non-transformable fonts cannot be used in character mode 3. If the font is not compatible with the character mode, the engine raises an error when the driver attempts to draw characters. The characteristics of the character modes are:

CM_MODE1 The position of characters after the first character is determined by the font metrics information. Character box, angle, shear, extra, break extra, and spacing are ignored.

CM_MODE2 The position of characters is determined by the font metrics information and the character attributes. Characters are not scaled, rotated, or sheared.

CM_MODE3 The position of characters is determined by the font metrics information and all character attributes. Characters can be scaled, rotated, and sheared.

Positioning is done by using the character reference point, defined within the font. When characters that are not hollow are drawn using an outline font they are filled using the character foreground color and mix.

GreLoadFont

```
(BOOL) GreLoadFont (pszFilename, pInstance, lFunction)
```

Loads fonts from a resource file. All fonts in the file are private, and are only available to the process that called this function.

Support

This function is supported by the graphics engine. It is available for use by the presentation driver at InnerGreEntry.

Stack Frame

(PSZ)	pszFilename	(pointer to a null-terminated string, containing path and name of the font file, identified by the file extension .FON)
(ULONG)	pInstance	(pointer to instance data)
(ULONG)	lFunction	(High-order word = Flags, Low-order word = NGreLoadFont)

Return Codes

This function returns BOOLEAN (fSuccess):

TRUE Successful.
FALSE Error.

When an error is detected, the graphics engine calls WinSetErrorInfo to post the condition. Reasons for failure of this function include:

PMERR_BASE_ERROR
PMERR_INSUFFICIENT_MEMORY
PMERR_INV_FONT_FILE_DATA.

```
(BOOL) GreUnLoadFont (pszFilename, pInstance, lFunction)
```

This function unloads the private font definitions previously loaded from the resource file indicated by pszFilename. Before unloading any fonts, the caller must ensure that:

1. The fonts are not in use for the current character set
2. The relevant lcids have been deleted (see "GreDeleteSetId" on page 15-37).

Support

GreUnLoadFont is supported by the graphics engine. It is available for use by the presentation driver at InnerGreEntry.

Stack Frame

(PSZ)	pszFilename	(pointer to a null-terminated string, containing the file path and name of the font file)
(ULONG)	pInstance	(pointer to instance data)
(ULONG)	lFunction	(High-order word = Flags, Low-order word = NGreUnLoadFont)

Return Codes

This function returns BOOLEAN (fSuccess):

TRUE Successful.
FALSE Error.

When an error is detected, the graphics engine calls WinSetErrorInfo to post the condition. Reasons for failure of this function include:

PMERR_BASE_ERROR
PMERR_FONT_FILE_NOT_LOADED.

GreQueryFonts

```
(LONG) GreQueryFonts (hdc, flOptions, pszFaceName, paMetrics, cMetricLen, pcFontCount, pInstance, lFunction)
```

This function returns details of fonts matching the face name addressed by pszFaceName.

Support

GreQueryFonts is supported by the graphics engine. It is available for use by the presentation driver at InnerGreEntry.

Stack Frame

(ULONG)	hdc	(device-context handle)
(ULONG)	flOptions	(flags indicating whether private fonts, public fonts, or both are required, see below)
(ULONG FAR *)	pszFaceName	(pointer to a string specifying the face name. When this is null, all faces are matched)
(ULONG FAR *)	paMetrics	(pointer to an array of font element records to which the metrics of matching fonts are returned, see below)
(ULONG)	cMetricLen	(length, in bytes, of each metrics structure in the paMetrics array)
(ULONG)	pcFontCount	(long pointer to cFontCount that specifies the number of fonts for which metrics are required. On return, this is updated to the number of fonts for which metrics were returned)
(ULONG)	pInstance	(pointer to instance data)
(ULONG)	lFunction	(High-order word = Flags, Low-order word = NGreQueryFonts)

flOptions These flags can be used in combination:

QFF_PUBLIC Enumerate public fonts.
QFF_PRIVATE Enumerate private fonts.

paMetrics This is a far pointer to an array of up to cFontCount font metric records, each of which contains a maximum of cMetricLen bytes. Note that for multi-codepage fonts the usCodePage field has no meaning and is set to zero.

Return Codes

This function returns the number of fonts not returned or GPI_ALTERROR when an error occurs.

When an error is detected, the graphics engine calls WinSetErrorInfo to post the condition. Reasons for failure of this function include:

PMERR_HDC_BUSY
PMERR_DEV_FUNC_NOT_INSTALLED
PMERR_COORDINATE_OVERFLOW
PMERR_INV_LENGTH_OR_COUNT
PMERR_INV_HDC
PMERR_INV_COORD_SPACE
PMERR_INV_OR_INCOMPAT_OPTIONS.

```
(BOOL) GreQueryFontAttributes (hdc, cbMetrics, pfmMetrics, pInstance, lFunction)
```

This function returns, at the location addressed by pfmMetrics, the metrics of the current font.

Support

GreQueryFontAttributes is supported by the graphics engine. It is available for use by the presentation driver at InnerGreEntry.

Stack Frame

(ULONG)	hdc	(device-context handle)
(ULONG)	cbMetrics	(size, in bytes, of the font metrics buffer)
(ULONG FAR *)	pfmMetrics	(long pointer to font metric block where the information is to be returned)
(ULONG)	pInstance	(pointer to instance data)
(ULONG)	lFunction	(High-order word = Flags, Low-order word = NGreQueryFontAttributes)

Return Codes

This function returns BOOLEAN (fSuccess):

TRUE Successful.
FALSE Error.

When an error is detected, the graphics engine calls WinSetErrorInfo to post the condition. Reasons for failure of this function include:

PMERR_HDC_BUSY
PMERR_DEV_FUNC_NOT_INSTALLED
PMERR_COORDINATE_OVERFLOW
PMERR_INV_SETID
PMERR_INV_LENGTH_OR_COUNT
PMERR_INV_HDC
PMERR_INV_COORD_SPACE.

GreQueryCodePageVector

```
pCodePage = GreQueryCodePageVector (ulCodePage, pInstance, lFunction)
```

Returns a long pointer to a vector, of 256 words, which is the code point to the glyph mapping number.

Support

This function is supported by the graphics engine. It is available for use by the presentation driver at InnerGreEntry.

Stack Frame

(ULONG)	ulCodePage	(code-page number)
(ULONG)	pInstance	(pointer to instance data)
(ULONG)	lFunction	(High-order word = Flags, Low-order word = NGreQueryCodePageVector)

Return Codes

This function returns a long pointer to the code-page vector or GPI_ERROR if an error occurred.

When an error is detected, the graphics engine calls WinSetErrorInfo to post the condition. Reasons for failure of this function include:

PMERR_INSUFFICIENT_MEMORY
PMERR_INV_CODEPAGE
PMERR_EXCEEDS_MAX_SEG_LENGTH.

```
cFonts = GreQueryFontFileDescriptions (pszFileName, pcFonts, paszNames, pInstance, lFunction)
```

Determines whether a file is a font file and, if true, returns the family and face names for the fonts in the file. The names are returned at the location addressed by paszNames.

The calling routine would usually call this function twice. The first call would have pcFonts set to zero and would determine the number of fonts in the file. Then, having allocated sufficient storage for the names, GreQueryFontFileDescriptions would be called again with pcFonts pointing to the font count.

Support

This function is supported by the graphics engine. It is available for use by the presentation driver at InnerGreEntry.

Stack Frame

(ULONG FAR *)	pszFileName	(long pointer to the file path and name of the font file)
(ULONG FAR *)	pcFonts	(pointer to a count of the maximum number of family and facename pairs to be returned. On completion, this is updated to the number of pairs actually returned)
(ULONG FAR *)	paszNames	(pointer to an array of 2×pcFonts 32-bit fields in which the family and facename pairs are returned. The family name is the first in each pair)
(ULONG)	pInstance	(pointer to instance data)
(ULONG)	lFunction	(High-order word = Flags, Low-order word = NGreQueryFontFileDescriptions)

Return Codes

This function returns the number of fonts not returned or GPI_ALTERROR if an error occurred.

When an error is detected, the graphics engine calls WinSetErrorInfo to post the condition. Reasons for failure of this function include:

PMERR_BASE_ERROR
PMERR_INV_FONT_FILE_DATA.

Journaling Functions

Journal functions allow presentation drivers to optimize the processing of data for raster devices, such as dot-matrix or laser printers, which print bands of data.

The size of a band is determined by the type of device and the amount of memory the driver can use to build its bit map. As an example, a color laser printer may need the full 24 bits per pel, in which instance several bands may be needed to make a page. A simple dot matrix printer that uses 1 bit per pel could treat the whole page as a single band.

The driver creates a journal file when the DC is enabled and starts recording in response to GreEscape (DEVESC_STARTDOC). The graphics engine stores the Gre... functions in the journal file as they are passed to the DC until told to stop recording by the driver. Typically, the driver processes the calls to produce the first band while the journal is being accumulated. When the journal file is complete, the driver passes the completed band to the spooler or the printer. It then plays the journal file and reprocesses the calls to produce the next band.

The entry points provided to allow journaling are:

GreCreateJournalFile

GreDeleteJournalFile

GreStartJournalFile

GreStopJournalFile

GrePlayJournalFile

GreOpenJournalFile.

```
hJournal = GreCreateJournalFile (pszFileName, flOption, cSize, pInstance, lFunction)
```

Creates a journal file on disk. The driver calls this function when responding to Enable (CompleteOpenDC).

Support

This function is supported by the graphics engine.

Stack Frame

(ULONG)	pszFileName	(pointer to a string containing the file name)
(ULONG)	flOption	(options, see below)
(ULONG)	cSize	(size, see below)
(ULONG)	pInstance	(pointer to instance data)
(ULONG)	lFunction	(High-order word = Flags, Low-order word = NGreCreateJournalFile)

flOptions Defined values are:

JNL_TEMP_FILE
An ordinary temporary journal file is created, pszFileName is ignored.

JNL_PERM_FILE
A permanent journal file is created, pszFileName points to a fully qualified path/file name.

JNL_ENGINERAM_FILE
A memory journal file is created in shared memory allocated by the engine, pszFileName is ignored.

JNL_USERRAM_FILE
A memory journal file is created in memory supplied by the caller. The location in memory is identified by the pointer passed in pszFileName.

JNL_DRAW_OPTIMIZATION
If set, optimization occurs, the Draw bit is not affected.

JNL_BOUNDS_OPTIMIZATION
If set, the Bounds bit is forced Off, otherwise, current behavior continues.

cSize If greater than zero, it is an indication as to how big the file needs to be. If flOption is JNL_USERRAM_FILE, cSize must be greater than zero and is the size of the buffer, which cannot be extended.

If cSize is zero, the calling routine does not know the size.

Return Codes

This function returns the journal handle (hJournal) or, if an error occurs, null.

If this function fails, the engine will set one of the following error codes:

PMERR_BASE_ERROR
PMERR_INV_IN_AREA
PMERR_INV_IN_PATH
PMERR_INSUFFICIENT_MEMORY
PMERR_INV_JOURNAL_OPTION
PMERR_RAM_JNL_FILE_TOO_SMALL.

GreDeleteJournalFile

```
(BOOL) = GreDeleteJournalFile (hJournal, pInstance, lFunction)
```

Deletes a journal file.

This function frees any objects associated with the journal file handle (such as compatible DCs, private clone regions or bit maps, and global memory segments). When the handle belongs to a temporary file the file is also deleted. Finally, the file handle itself is freed.

Support

This function is supported by the graphics engine.

Stack Frame

(ULONG)	hJournal	(journal file handle)
(ULONG)	pInstance	(pointer to instance data)
(ULONG)	lFunction	(High-order word = Flags, Low-order word = NGreDeleteJournalFile)

Return Codes

This function returns BOOLEAN (fSuccess):

TRUE Successful.
FALSE Error.

When an error is detected, the graphics engine calls WinSetErrorInfo to post the condition. Reasons for failure of this function include:

PMERR_BASE_ERROR
PMERR_INV_IN_AREA
PMERR_INV_IN_PATH
PMERR_HDC_BUSY
PMERR_DEV_FUNC_NOT_INSTALLED
PMERR_COORDINATE_OVERFLOW
PMERR_INV_PRIMITIVE_TYPE
PMERR_INV_SETID
PMERR_INV_CHAR_SET_ATTR
PMERR_INV_MARKER_SET_ATTR
PMERR_INV_PATTERN_SET_ATTR
PMERR_INV_MARKER_SYMBOL_ATTR
PMERR_UNSUPPORTED_ATTR
PMERR_INV_MIX_ATTR
PMERR_INV_BACKGROUND_MIX_ATTR
PMERR_INV_PATTERN_REF_PT_ATTR
PMERR_INV_LINE_WIDTH_ATTR
PMERR_INV_GEOM_LINE_WIDTH_ATTR
PMERR_INV_LINE_TYPE_ATTR
PMERR_INV_LINE_JOIN_ATTR
PMERR_INV_LINE_END_ATTR
PMERR_INV_CHAR_MODE_ATTR
PMERR_INV_CHAR_SHEAR_ATTR
PMERR_INV_CHAR_DIRECTION_ATTR
PMERR_UNSUPPORTED_ATTR_VALUE
PMERR_INSUFFICIENT_MEMORY
PMERR_INV_LENGTH_OR_COUNT
PMERR_INV_CODEPAGE
PMERR_EXCEEDS_MAX_SEG_LENGTH
PMERR_INV_RECT
PMERR_INV_COORDINATE
PMERR_INV_HDC
PMERR_INV_HRGN
PMERR_HRGN_BUSY

PMERR_INV_METAFILE
PMERR_JFILE_BUSY
PMERR_INV_HJOURNAL
PMERR_REGION_IS_CLIP_REGION
PMERR_INV_REGION_CONTROL
PMERR_INV_HBITMAP
PMERR_INV_DC_TYPE
PMERR_INV_COORD_SPACE
PMERR_INV_ID
PMERR_BITMAP_IS_SELECTED
PMERR_HBITMAP_BUSY
PMERR_INV_INFO_TABLE
PMERR_INV_BITMAP_DIMENSION
PMERR_INV_SCAN_START
PMERR_INV_PATTERN_SET_FONT
PMERR_HUGE_FONTS_NOT_SUPPORTED
PMERR_INV_COLOR_ATTR
PMERR_INV_BACKGROUND_COL_ATTR.

GreStartJournalFile

```
(BOOL) = GreStartJournalFile (hdc, hJournal, pInstance, lFunction)
```

Starts the journaling process. Opens the previously created journal file and turns on the COM_RECORDING bit. Subsequent calls to this DC drop through AccumulateJournalFile until StopJournalFile is called.

Support

This function is supported by the graphics engine. It is available for use by the presentation driver at InnerGreEntry.

Stack Frame

(ULONG)	hdc	(device context handle)
(ULONG)	hJournal	(journal file handle)
(ULONG)	pInstance	(pointer to instance data)
(ULONG)	lFunction	(High-order word = Flags, Low-order word = NGreStartJournalFile)

Return Codes

This function returns BOOLEAN (fSuccess):

TRUE Successful.
FALSE Error.

When an error is detected, the graphics engine calls WinSetErrorInfo to post the condition. Reasons for failure of this function include:

PMERR_BASE_ERROR
PMERR_INV_IN_AREA
PMERR_INV_IN_PATH
PMERR_HDC_BUSY
PMERR_DEV_FUNC_NOT_INSTALLED
PMERR_COORDINATE_OVERFLOW
PMERR_INV_PATTERN_SET_ATTR
PMERR_INV_MIX_ATTR
PMERR_INV_BACKGROUND_MIX_ATTR
PMERR_INV_PATTERN_REF_PT_ATTR
PMERR_INV_LINE_TYPE_ATTR
PMERR_INV_CHAR_MODE_ATTR
PMERR_INV_CHAR_DIRECTION_ATTR
PMERR_INSUFFICIENT_MEMORY
PMERR_INV_LENGTH_OR_COUNT
PMERR_INV_CODEPAGE
PMERR_EXCEEDS_MAX_SEG_LENGTH
PMERR_INV_RECT
PMERR_INV_COORDINATE
PMERR_INV_HDC
PMERR_INV_HRGN
PMERR_INV_METAFILE
PMERR_JFILE_BUSY
PMERR_INV_HJOURNAL
PMERR_INV_DC_DATA
PMERR_INV_REGION_CONTROL
PMERR_INV_DRIVER_NAME
PMERR_RAM_JNL_FILE_TOO_SMALL
PMERR_INV_HBITMAP
PMERR_INV_DC_TYPE
PMERR_INV_COORD_SPACE
PMERR_INV_ID
PMERR_BITMAP_IS_SELECTED
PMERR_HBITMAP_BUSY

PMERR_INV_INFO_TABLE
PMERR_INV_BITMAP_DIMENSION
PMERR_INV_SCAN_START
PMERR_INV_PATTERN_SET_FONT
PMERR_HUGE_FONTS_NOT_SUPPORTED
PMERR_INV_COLOR_ATTR
PMERR_INV_BACKGROUND_COL_ATTR.

GreStopJournalFile

```
(BOOL) = GreStopJournalFile (hdc, hJournal, pInstance, lFunction)
```

Writes the END_OF_JOURNALFILE marker into the journal file, closes the journal file, and turns off the COM_RECORDING bit.

Support

This function is supported by the graphics engine. It is available for use by the presentation driver at InnerGreEntry.

Stack Frame

(ULONG)	hdc	(device context handle)
(ULONG)	hJournal	(journal file handle)
(ULONG)	pInstance	(pointer to instance data)
(ULONG)	lFunction	(High-order word = Flags, Low-order word = NGreStopJournalFile)

Return Codes

This function returns BOOLEAN (fSuccess):

TRUE Successful.
FALSE Error.

When an error is detected, the graphics engine calls WinSetErrorInfo to post the condition. Reasons for failure of this function include:

PMERR_BASE_ERROR
PMERR_INV_IN_AREA
PMERR_INV_IN_PATH
PMERR_INV_HDC
PMERR_INV_METAFILE
PMERR_JFILE_BUSY
PMERR_INV_HJOURNAL
PMERR_INV_DC_DATA
PMERR_RAM_JNL_FILE_TOO_SMALL.

```
(BOOL) = GrePlayJournalFile (hdc, hJournal, pInstance, lFunction)
```

This function plays a journal file to the specified DC. The journal file is read into memory and each journaled call is played.

Each journaled record is processed before playing to the extent that long pointers to data are fixed up and clone objects (regions or bit maps) are created, if necessary, from the journaled data.

It is assumed that any single journaled function and associated data fits in a 32KB buffer. If the journaled record contains region rectangles or bit-map bits, these do not count in this restriction.

Support

This function is supported by the graphics engine. It is available for use by the presentation driver at InnerGreEntry.

Stack Frame

(ULONG)	hdc	(device context handle)
(ULONG)	hJournal	(journal file handle)
(ULONG)	pInstance	(pointer to instance data)
(ULONG)	lFunction	(High-order word = Flags, Low-order word = NGrePlayJournalFile)

Return Codes

This function returns BOOLEAN (fSuccess):

TRUE Successful.
FALSE Error.

When an error is detected, the graphics engine calls WinSetErrorInfo to post the condition. Reasons for failure of this function include:

PMERR_BASE_ERROR
PMERR_INV_IN_AREA
PMERR_INV_IN_PATH
PMERR_HDC_BUSY
PMERR_DEV_FUNC_NOT_INSTALLED
PMERR_COORDINATE_OVERFLOW
PMERR_BITMAP_NOT_SELECTED
PMERR_INV_PATTERN_SET_ATTR
PMERR_INV_MIX_ATTR
PMERR_INV_BACKGROUND_MIX_ATTR
PMERR_INV_PATTERN_REF_PT_ATTR
PMERR_INV_LINE_TYPE_ATTR
PMERR_INV_CHAR_MODE_ATTR
PMERR_INV_CHAR_DIRECTION_ATTR
PMERR_INV_REGION_MIX_MODE
PMERR_INSUFFICIENT_MEMORY
PMERR_INV_LENGTH_OR_COUNT
PMERR_INV_CODEPAGE
PMERR_EXCEEDS_MAX_SEG_LENGTH
PMERR_INV_RECT
PMERR_INV_COORDINATE
PMERR_INV_HDC
PMERR_INV_HRGN
PMERR_HRGN_BUSY
PMERR_INV_METAFILE
PMERR_JFILE_BUSY
PMERR_INV_HJOURNAL
PMERR_INV_DC_DATA
PMERR_REGION_IS_CLIP_REGION

PMERR_INV_REGION_CONTROL
PMERR_INV_DRIVER_NAME
PMERR_RAM_JNL_FILE_TOO_SMALL
PMERR_INV_HBITMAP
PMERR_INV_DC_TYPE
PMERR_INV_COORD_SPACE
PMERR_INV_ID
PMERR_BITMAP_IS_SELECTED
PMERR_HBITMAP_BUSY
PMERR_INV_USAGE_PARM
PMERR_INV_INFO_TABLE
PMERR_INV_BITMAP_DIMENSION
PMERR_INCORRECT_DC_TYPE
PMERR_INV_SCAN_START
PMERR_INV_PATTERN_SET_FONT
PMERR_HUGE_FONTS_NOT_SUPPORTED
PMERR_INV_COLOR_ATTR
PMERR_INV_BACKGROUND_COL_ATTR.

```
hJournal = GreOpenJournalFile (pszFileName, flOption, cBufSize, pInstance, lFunction)
```

Opens a journal file for play. The journal file must exist and must contain valid journaled calls.

Support

This function is supported by the graphics engine. It is available for use by the presentation driver at InnerGreEntry.

Stack Frame

(ULONG FAR *)	pszFileName	(pointer a string containing the file name)
(ULONG)	flOption	(see below)
(ULONG)	cBufSize	(size of buffer required for the file)
(ULONG)	pInstance	(pointer to instance data)
(ULONG)	lFunction	(High-order word = Flags, Low-order word = NGreOpenJournalFile)

flOption An option flag that identifies the type of journal file. Valid values are:

JNL_PERM_FILE A disk file, with the name pszFileName, that was created using GreCreateJournalFile(JNL_PERM_FILE).

Bit 3 A shared memory file pointed to by pszFileName and already filled with complete journal records. This journal file might have been created using GreCreateJournalFile(JNL_USERRAM_FILE) and filled by the engine or it may have been produced in some other way.

Return Codes

The handling routine returns hJournal or, if an error occurs, null.

When an error is detected, the graphics engine calls WinSetErrorInfo to post the condition. Reasons for failure of this function include:

PMERR_BASE_ERROR
PMERR_INV_IN_AREA
PMERR_INV_IN_PATH
PMERR_INSUFFICIENT_MEMORY
PMERR_INV_JOURNAL_OPTION
PMERR_RAM_JNL_FILE_TOO_SMALL.

Chapter 16. System Functions

This chapter describes system functions that are available to presentation drivers running at ring 2:

- FSRSemCheck
- FSRSemEnter
- FSRSemExit
- FSRSemLeave
- GetDriverInfo
- InnerGreEntry
- SetDriverInfo
- SSAllocHuge
- SSAllocSeg
- SSFreeSeg
- VioGetPSAddress
- VisRegionCallBack
- WinSetErrorInfo.

FSRSemCheck

```
ULONG APIENTRY FSRSemCheck (npFSRSem)
```

This function checks the ownership of the FSR semaphore addressed by npFSRSem.

FSRSemCheck is available across the system at ring 2; presentation drivers do not need to import this function through the module definition file.

Parameters

npFSRSem *(PFSRSEM)*

Near pointer to an FSRSEM structure. (See "FSRSemEnter" on page 16-3.)

Return Codes

FSRSemCheck returns a ULONG value (DX:AX) that defines the current owner of the semaphore. The high-order portion in DX shows the owning process, the low-order portion in AX shows the owning thread. Possible values are:

DX:AX = NULL	Owner is calling process and thread.
DX = NULL, AX ≠ NULL	Owner is calling process, AX contains thread ID.
DX:AX = FFFFFFFFH	Semaphore has no owner.
Any other value	DX contains ID of owning process, AX contains thread ID.

```
INT APIENTRY FSRSemEnter (npFSRSem)
```

This function attempts to acquire ownership of a FSR semaphore for the calling process and thread. If the semaphore is already owned by the calling process and thread, the system increments the semaphore's usage counter and returns Success.

FSRSemEnter is available across the system at ring 2; presentation drivers do not need to import this function through the module definition file.

Parameters

npFSRSem *(PFSRSEM)*

Near pointer to an FSRSEM structure. The only field that the presentation driver should access directly is usClient; all other fields are reserved for use by the system. For details, see "Remarks" below for details.

Return Codes

FSRSemCheck returns an integer value:

0 Success.
—1 Fail.

Remarks

The internal FSRSem... calls provide semaphore facilities using a FSRSEM structure located in the presentation driver's data segment. This structure, which must be defined by the presentation driver, contains the following fields:

usLength Length of structure.
usPadding
usProcessID ID of owning process.
usThreadID ID of owning thread.
usUsage Usage counter.
usClient Client field, see Note.
ulTimeout System default timeout value.
ulRamSem Semaphore.

Note: The usClient field is the only field that, for normal processing, the presentation driver should access directly. This field is initialized as 0 when the semaphore is first acquired and thereafter is controlled by the driver. The intention is to give a DC some way of flagging a semaphore to notify the driver's ExitList processing routine of any special action it needs to take when releasing the semaphore.

Presentation drivers would normally keep a table of FSRSEM structures with one for each type of resource that they use. The example on the following page shows a typical definition of a FSRSEM structure and table.

```
typedef struct _FSRSEM {  /* fsrs */
    USHORT Length;
    USHORT Padding;
    USHORT ProcID;
    USHORT ThrdID;
    USHORT Usage;
    USHORT Client;
    ULONG  Timeout;
    ULONG  RAMsem;
} FSRSEM;
typedef FSRSEM *PFSRSEM;
   :
FSRSEM RAMSemaphoreTable[RAM_SEMAPHORE_TABLE_SIZE] =
    { sizeof(FSRSEM), 0, 0, 0, 0, 0, -1L, 0L,
      sizeof(FSRSEM), 0, 0, 0, 0, 0, -1L, 0L,
      sizeof(FSRSEM), 0, 0, 0, 0, 0, -1L, 0L,
      sizeof(FSRSEM), 0, 0, 0, 0, 0, -1L, 0L,
      sizeof(FSRSEM), 0, 0, 0, 0, 0, -1L, 0L,
      sizeof(FSRSEM), 0, 0, 0, 0, 0, -1L, 0L,
      sizeof(FSRSEM), 0, 0, 0, 0, 0, -1L, 0L,
      sizeof(FSRSEM), 0, 0, 0, 0, 0, -1L, 0L,
      sizeof(FSRSEM), 0, 0, 0, 0, 0, -1L, 0L,
      sizeof(FSRSEM), 0, 0, 0, 0, 0, -1L, 0L
};

PFSRSEM    lpCreateHeapSem     = (PFSRSEM) &RAMSemaphoreTable[0];
PFSRSEM    lpGlobalHeapSem     = (PFSRSEM) &RAMSemaphoreTable[1];
PFSRSEM    lpJournalNameSem    = (PFSRSEM) &RAMSemaphoreTable[2];
PFSRSEM    pBMLockSem             = (PFSRSEM) &RAMSemaphoreTable[3];


/*****************************************************************************/
/*  Device semaphores table                                                  */
/*  If additional devices are added to this table then the definition */
/*  of NumberOfPDevices must be updated.;;                                   */
/*****************************************************************************/
DeviceSemaphoreTableEntry DevSemTable[] =
    { "LPT1",                              /* device name       */
      (PFSRSEM) &RAMSemaphoreTable[4],     /* control semaphore */
      "LPT2",
      (PFSRSEM) &RAMSemaphoreTable[5],
      "LPT3",
      (PFSRSEM) &RAMSemaphoreTable[6],
      "COM1",
      (PFSRSEM) &RAMSemaphoreTable[7],
      "COM2",
      (PFSRSEM) &RAMSemaphoreTable[8],
      "COM3",
      (PFSRSEM) &RAMSemaphoreTable[9],
      "PRN",
      (PFSRSEM) &RAMSemaphoreTable[4] };

DeviceSemaphoreTable lpDevSemTable =
                     (DeviceSemaphoreTableEntry far *) DevSemTable;

DeviceSemaphoreTableEntry DevSemQueuedDCTable =
    { FNULL,
      (PFSRSEM) &RAMSemaphoreTable[10] };
```

```
VOID APIENTRY FSRSemExit (npFSRSem)
```

This function checks the ownership of the FSR semaphore addressed by npFSRSem and, if it is owned by the calling process and thread, resets the usage count and releases the semaphore. Presentation drivers that share resources between different instances of a DC should first call FSRSemCheck to see if the semaphore is owned by the process and thread that opened the calling DC.

FSRSemExit is available across the system at ring 2; presentation drivers do not need to import this function through the module definition file.

Parameters

npFSRSem *(PFSRSEM)*

Near pointer to an FSRSEM structure. (See "FSRSemEnter" on page 16-3.)

Return Codes

None, FSRSemExit is a VOID function.

FSRSemLeave

```
VOID APIENTRY FSRSemLeave (npFSRSem)
```

This function checks the ownership of the FSR semaphore addressed by npFSRSem and, if it is owned by the calling process and thread, decrements the usage count for the semaphore; if the usage count goes to zero, the system releases the semaphore. Presentation drivers that share resources between different instances of a DC should first call FSRSemCheck to see if the semaphore is owned by the process and thread that opened the calling DC.

FSRSemExit is available across the system at ring 2; presentation drivers do not need to import this function through the module definition file.

Parameters

npFSRSem *(PFSRSEM)*

Near pointer to an FSRSEM structure. (See "FSRSemEnter" on page 16-3.)

Return Codes

None, FSRSemExit is a VOID function.

```
LONG APIENTRY GetDriverInfo (hEngObject, ulIndex, hdc)
```

This function returns a driver-object handle for the engine object identified by hEngObject. The parameter hEngObject can be a DC handle or an engine bit-map handle such as the source handle that would be passed to the drivers GreBitblt routine. If hEngObject is a DC, the return code is a pointer (pInstance) to that DC's instance data; if hEngObject is a bit map, the return code is the driver's handle for the bit map (that is, the handle returned to the driver when the bit map was created).

GetDriverInfo includes a check to ensure that the object is, or was created by, an instance of a DC for the same device as the DC identified by hdc. If this is not true, the function returns GPI_ALTERROR.

GetDriverInfo is exported by the graphics engine at ordinal 30 and, if the driver wants to call it, it must be imported by the driver's module definition file.

Parameters

hEngObject *(ULONG)*
The handle of a DC or a bit map.

ulIndex *(ULONG)*

Index = 0 hEngObject is a DC handle.
Index = 1 hEngObject is a bit-map handle.

hdc *(ULONG)*
Device context handle (identifies the calling DC).

Return Codes

If successful the function returns the instance pointer associated with the object DC or the driver's handle to the object bit map. If an error is detected, the function returns GPI_ALTERROR.

InnerGreEntry

```
ULONG APIENTRY InnerGreEntry (lArgs, pInstance, lFunction)
```

InnerGreEntry is a general entry point to the functions (simulations of PDI functions and internal functions) that are supported in the graphics engine. This entry point is exported by the graphics engine (PMGRE) at ordinal 3; to use it, the presentation driver would usually import the entry point by ordinal and assign a meaningful name to it. For details, see "Imported Functions" on page 10-4.

Parameters

lArgs *(ULONG)*
This is a list of the arguments, excluding pInstance and lFunction, that are required for the function being called. The format of the list is:

```
hdc, parm1, parm2, ....., parmN
```

pInstance *(ULONG)*
This parameter must be null.

lFunction *(ULONG)*
Flags and function number. The high-order word of lFunction contains the flags; the low-order word contains the function number.

Named constants are defined in PMDDI.H for the function numbers. For example, NGrePolyFillet is used to call the simulation of GrePolyFillet in the graphics engine.

The following flags can be set in calls made through InnerGreEntry:

FUNBIT_HAS_HDC	The first parameter is a device context handle (hdc).
FUNBIT_GPIDISPATCH	This flag indicates that the pInstance parameter should be copied to the graphic engine's return address, this ensures that the results of the function are returned to the correct instance of the presentation driver.

Return Codes

The return codes are those returned by the Gre... function identified by the function number in lFunction.

```
LONG APIENTRY SetDriverInfo (hDrvObject, hEngObject, ulIndex, hdc)
```

This function associates a driver-object handle with the object identified by an engine-object handle. Object handles are stored in the in bit maps when they are created, calling SetDriverInfo provides a way in which the presentation driver can change the driver handle in the bit-map.

SetDriverInfo includes a check to ensure that the object is, or was created by, an instance of a DC for the same device as the DC identified by hdc. If this is not true, the function returns GPI_ALTERROR

SetDriverInfo is exported by the graphics engine at ordinal 31 and, if a driver wants to call it, it must be imported by the driver's module definition file.

Parameters

hDrvObject *(ULONG)*
The driver's handle for the object (this value must not be −1).

hEngObject *(ULONG)*
The Engine's handle for the object. must not be −1).

lIndex *(ULONG)*

lIndex = 0 Reserved.
lIndex = 1 hObject is a bit-map handle.

hdc *(ULONG)*
Device context handle.

Return Codes

If successful the function returns the driver-object handle that was associated with the object before any changes made by SetDriverInfo. If an error is detected, the function returns GPI_ALTERROR.

SSAllocHuge

```
USHORT APIENTRY SSAllocHuge (usNumSeg, usPartialSeg, psel, usMaxNumSeg)
```

This function allocates multiple segments of shared memory that are managed by the selector server component of the graphics engine. Presentation drivers for display devices should use SSAllocHuge to allocate memory for objects such as bit maps and regions. This is necessary to ensure that the returned selector is a global selector for which the system can set a new owner or which can be marked as having no owner.

SSAllocHuge is available across the system at ring 2; presentation drivers do not need to import this function through the module definition file.

Parameters

usNumSeg *(USHORT)*
Number of 64KB segments.

usPartialSeg *(USHORT)*
Number of bytes in last segment.

psel *(PSEL)*
Pointer to a SEL data type in which the function returns the memory selector.

usMaxNumSeg *(USHORT)*
Maximum number of 64KB segments.

Return Codes

SSAllocHuge returns a USHORT value:

NO_ERROR
ERROR_NOT_ENOUGH_MEMORY
ERROR_INVALID_PARAMETER.

```
USHORT APIENTRY SSAllocSeg (usSize, psel)
```

This function allocates a segment of shared memory that is managed by the selector server component of the graphics engine. Presentation drivers for display devices should use SSAllocSeg to allocate memory for objects such as bit maps and regions. This is necessary to ensure that the returned selector is a global selector for which the system can set a new owner or which can be marked as having no owner.

SSAllocSeg is available across the system at ring 2; presentation drivers do not need to import this function through the module definition file.

Parameters

usSize *(USHORT)*
Number of bytes required.

psel *(PSEL)*
Pointer to a SEL data type in which the function returns the memory selector.

Return Codes

SSAllocSeg returns a USHORT value:

NO_ERROR
ERROR_NOT_ENOUGH_MEMORY
ERROR_INVALID_PARAMETER.

SSFreeSeg

```
USHORT APIENTRY SSFreeSeg (psel)
```

This function frees a segment of shared memory that was allocated by a call to SSAllocSeg or SSAllocHuge.

SSFreeSeg is available across the system at ring 2; presentation drivers do not need to import this function through the module definition file.

Parameters

psel *(PSEL)*
Pointer to the memory selector.

Return Codes

SSFreeSeg returns a USHORT value:

NO_ERROR
ERROR_ACCESS_DENIED
ERROR_INVALID_PARAMETER.

```
USHORT APIENTRY VioGetPSAddress (pps, hvio)
```

This function returns a pointer to the VioPresentationSpace structure for the AVIO presentation space identified by hvio. The function is used only by presentation drivers for display devices; these drivers need to call VioGetPSAddress from their GreDeviceSetAVIOFont routine (see page 13-9).

VioGetPSAddress is exported by VIOCALLS.DLL. Presentation drivers for display devices import it by including the following statement in the IMPORTS section of the module definition file:

```
VioGetPSAddress=VIOCALLS.VioGetPSAddress
```

Note: VioGetPSAddress runs at ring (privilege level) 3, the presentation driver must, therefore, use DosCallBack to make the transition from ring 2 to ring 3. For details, see the *Programming Reference*.

Parameters

pps *(VioPresentationSpace FAR)*
A long pointer to an address in which the pointer to the VioPresentationSpace structure is to be returned.

hvio *(HVIO)*
AVIO presentation-space handle.

Return Codes

This function returns a USHORT value:

NO_ERROR Successful
ERROR_VIO_INVALID_HANDLE Invalid handle
ERROR_VIO_NOT_PRES_MGR_SG Wrong screen group.

Remarks

Because DosCallBack only passes a pointer to a function, a strategy is needed for the driver to pass the handle to ring 3 and to receive the returned pointer. A typical presentation driver has a ring 3 routine which it uses to call VioGetPSAddress. The driver, running at ring 2, calls DosAllocSeg to create a shareable stack and passes the handle (hvio) on to it. It then calls the ring 3 routine using the DosCallBack function. The ring 3 routine takes the handle from the stack and calls VioGetPSAddress. The pointer is then returned on the stack.

VisRegionCallBack

```
BOOL APIENTRY VisRegionCallBack (hdc)
```

This function recalculates the visible region and validates the DC region. The handling routine in the graphics engine does the necessary calculations and calls the GreNotifyClipChange (page 12-69) routine in the presentation driver so that the driver can get the new clip regions.

VisRegionCallBack is used only by presentation drivers for display devices. These drivers maintain an HDC_IS_DIRTY flag that is set on by GreDeviceInvalidateClipRegion and cleared by GreNotifyClipChange. In the driver, the handling routines for all drawing functions should check the HDC_IS_DIRTY flag and, if it is set, call VisRegionCallBack before drawing.

VisRegionCallBack is available across the system at ring 2; presentation drivers do not need to import this function through the module definition file.

Parameters

hdc *(ULONG)*
Device context handle.

Return Codes

This function returns BOOLEAN (fSuccess):

TRUE Successful.
FALSE Error.

```
void cdecl FAR WinSetErrorInfo (idError, fsOptions, arg1, ..., argN)
```

This function posts an error message that can be retrieved by a program, running at ring 3, calling WinGetLastError. The syntax of the call to WinSetErrorInfo allows a variable number of parameters; the minimum requirement is the appropriate ERRORID structure and a null value for fsOptions.

WinSetErrorInfo is available across the system at ring 2; presentation drivers do not need to import this function through the module definition file.

idError *(ERRORID)*
This is an ERRORID structure:

lSeverity Valid values are:
SEVERITY_NOERROR
SEVERITY_WARNING
SEVERITY_ERROR
SEVERITY_SEVERE
SEVERITY_UNRECOVERABLE.

lErrorCode Error code. See "Presentation Manager Error Codes" on page 10-13 for a list of defined values.

fsOptions *(USHORT)*
Option flags:

4000H Do not call DosBeep.
2000H Do not prompt the user.
0008H The next parameter is a base OS/2 error code.

arg1...argN *(USHORT)*
Variable number of optional arguments. These parameters have no significance except when the presentation driver is reporting a base OS/2 error, see below.

Return Codes

This function returns VOID.

Remarks

The two examples, below, show how WinSetErrorInfo is used to post a presentation driver error and to post a base OS/2 error.

```
#define   NOBEEP   0x4000      /* Define the flags. */
#define   NOPROMP  0x2000
#define   DOSERROR 0x0008

/***  To post a presentation driver error  ***/

WinSetErrorInfo ( ERRORID (SEVERITY_ERROR, PMERR_INV_COORDINATE), NOBEEP );

/***  To post a base OS/2 error  ***/

WinSetErrorInfo ( ERRORID (SEVERITY_WARNING, ErrorCode), (DOSERROR+NOBEEP+NOPROMPT), DosErrorCode );
```

Chapter 17. Presentation Manager Spooler

For the Presentation Manager, hard-copy devices, such as printers and plotters, are queued devices. When an application writes to one of these devices, the presentation driver creates a spool file and writes the data to that file. The data is printed when it is complete and the required device is free.

Two instances of a device context are required to support queued data. The first instance is opened as an OD_QUEUED device by the application program; this DC buffers the data, does any processing that is required, and then writes the data to a spool file. The second instance is opened as an OD_DIRECT device by the queue processor; this DC receives data from the spool file, does any processing that is required, and, using the Prt... interface, sends the data to the kernel device driver.

Spooler Components

The spooler interface (Spl...) is implemented in two libraries, PMSPL.DLL and PMPRINT.QPR, that support four activities:

1. Managing spool buffers for PM_Q_STD data (SplStd... interface)
2. Managing queued files (SplQm... interface)
3. Processing queued files (SplQp... interface)
4. Displaying messages on the screen (SplMesageBox).

Spool File Creation

The steps taken by the presentation driver to create a spool file are determined by the data type for which the DC was enabled. All presentation drivers must support PM_Q_STD and PM_Q_RAW; overviews of creating a spool file for these data types are shown below.

You can define your own data type. If you do, you should choose a name that is not likely to clash with other user-defined data types; the name must be a string of up to 16 characters in the ranges of 'A' – 'Z', '0' – '9', and '_'. And be aware that the data type will only be of use to applications that know about it and with presentation drivers that implement it.

PM_Q_STD

PM_Q_STD data is a file of Gpi... calls that describe the output document. This data type is independent of the device and the presentation driver.

The presentation driver invokes the spooler's standard interface (SplStd...); all subsequent Gpi... calls and some escape codes sent to the DC are recorded in a spool buffer. When DEVESC_ENDDOC is detected, the presentation driver ends the recording and invokes the spooler's queue manager interface (SplQm...) to write the buffered data into a spool file. The sequence of events is as follows:

Component	Action
Application	Calls DevOpenDC to open an OD_QUEUED device for printing PM_Q_STD data.
Driver	Calls SplStdOpen to open a spool buffer.
Application	Calls DevEscape with DEVESC_STARTDOC.
Driver	Calls SplStdStart to start recording.
Spooler	Records all Gpi... calls and some escape codes in the spool buffer.
	Note: Recording does not stop the flow of Gpi... calls through the system; these calls are processed and the resulting Gre... calls are passed on to handling routines in the graphics engine and presentation driver. Special considerations apply for escape codes; for details see the individual escape codes under "GreEscape" on page 12-115.
Driver	Behaves as if the DC was opened as OD_INFO. The presentation driver tracks the current position, does any bounds calculation required, and responds to queries from the application. In particular, the presentation driver must be able to understand and reply to: DevQueryCaps DevQueryHardCopyCaps DevQueryDeviceNames DevPostDeviceModes.

	The calls can be journaled but the journal file is not saved.
Application	Calls DevEscape with DEVESC_ENDDOC.
Driver	Calls SplStdStop to stop the recording.
Driver	Calls SplQmOpen and SplQmStartDoc to open and start a spool file.
Driver	Calls SplStdGetBits to get data from the spool buffer into memory that is owned by the presentation driver. (The driver may need to loop on this step and the next if the spooled data is larger than the available memory.)
Driver	Calls SplQmWrite to write the data in the spool file.
Driver	Calls SplStdDelete to delete the data in the spool buffer.
Driver	Calls SplQmEndDoc to stop the spool file.
Application	Calls DevCloseDC.
Driver	Calls SplStdClose and SplQmClose to close the spool buffer and the spool file.

PM_Q_RAW

PM_Q_RAW data is a device-dependent bit stream.

Component	Action
Application	Calls DevOpenDC to open an OD_QUEUED device for printing PM_Q_RAW data.
Driver	Calls SplQmOpen to open a spool file.
Driver	Calls GreCreateJournalFile to create a journal file.
Application	Calls DevEscape with DEVESC_STARTDOC to start the document.
Driver	Calls SplQmStartDoc to start the spool file.
Driver	Calls GreStartJournalFile to start recording Gre... calls
	Note: Recording does not block the flow of Gre... calls to handling routines in the graphics engine and presentation driver.
Driver	Processes the incoming Gre... calls to create the first band of data for the spool file.
Application	Calls DevEscape with DEVESC_ENDDOC to end the document.
Driver	Calls GreStopJournalFile to stop the journal file.
Driver	Calls SplQmWrite to write the first band in the spool file.
Driver	Calls GrePlayJournalFile to play the journal file.
Driver	Processes each Gre... call to create the second band of data for the spool file.
Driver	Calls SplQmWrite to write the second band in the spool file.
Driver	Plays the journal file repeatedly until all the bands have been processed and passed to the spooler.
Driver	Calls GreDeleteJournalFile.
Driver	Calls SplQmEndDoc and returns the job ID to the application's Escape(DEVESC_ENDDOC).
Application	Calls DevCloseDC.
Driver	Calls SplQmClose.

Spooler Functions

The following functions are available to do file functions on a spool buffer containing PM_Q_STD data:

SplStdClose

SplStdDelete

SplStdGetBits

SplStdStart

SplStdStop

SplStdOpen

SplStdQueryLength.

SplStdClose

```
BOOL APIENTRY SplStdClose (hdc)
```

This function closes the PM_Q_STD buffer.

The call to SplStdClose is made from the presentation driver's BeginCloseDC routine when the device type is OD_QUEUED and the data type is PM_Q_STD. SplStdClose must not be called at any other time.

Parameters

hdc *(HDC)*
Device context handle.

Return Codes

This function returns BOOLEAN (fSuccess):

TRUE Successful.
FALSE Error.

```
BOOL APIENTRY SplStdDelete (hstd)
```

This function deletes the PM_Q_STD buffer identified by hstd. Any data in the buffer is lost.

Parameters

hstd *(HSTD)*
Handle to PM_Q_STD data.

Return Codes

This function returns BOOLEAN (fSuccess):

TRUE Successful.
FALSE Error.

SplStdGetBits

```
BOOL APIENTRY SplStdGetBits (hstd, iStart, cBytes, pAddress)
```

This function transfers data from the identified PM_Q_STD buffer into a buffer owned by the presentation driver.

Before calling SplStdGetBits the presentation driver should call SplStdQueryLength to determine the length of the PM_Q_STD data. Then, depending upon the length, the driver should allocate a buffer big enough to contain all of the data or allocate a smaller buffer and get the data in a series of calls to SplStdGetBits.

Parameters

hstd *(HSTD)*
Handle to the PM_Q_STD buffer.

iStart *(LONG)*
Offset to the byte at which transfer must start. Used when the data is got in a series of calls to SplStdGetBits.

cBytes *(LONG)*
Number of bytes to transfer.

pAddress *(PCH)*
Pointer to the presentation driver's data buffer.

Return Codes

This function returns BOOLEAN (fSuccess):

TRUE Successful.
FALSE Error.

```
BOOL APIENTRY SplStdStart (hdc)
```

This function starts the recording of GPI and DevEscape calls as a metafile in the PM_Q_STD buffer. Note that the calls are still processed by the system and passed on to the graphics engine and presentation driver as calls to the relevant Gre... functions.

The call to SplStdStart is usually made from the presentation driver's Escape routine when DEVESC_STARTDOC is received. Some early drivers made this call from the CompleteOpenDC routine to accommodate applications that did not call DevEscape(DEVESC_STARTDOC) at the start of the document.

Parameters

hdc *(HDC)*
Device context handle.

Return Codes

This function returns BOOLEAN (fSuccess):

TRUE Successful.
FALSE Error.

SplStdStop

```
HSTD APIENTRY SplStdStop (hdc)
```

This function stops the recording of GPI and DevEscape calls in the PM_Q_STD buffer. The call to SplStdStop is made from the presentation driver's Escape routine when DEVESC_ENDOC is received.

Parameters

hdc *(HDC)*

Device context handle.

Return Codes

If successful, SplStdStop returns the handle to the buffer (hstd) that contains the recorded GPI and DevEscape data. If an error occurs, the function returns SPL_ERROR.

```
BOOL APIENTRY SplStdOpen (hdc)
```

This function opens a spool buffer for PM_Q_STD data.

The call to SplStdOpen is made from the presentation driver's CompleteOpenDC routine when the device type is OD_QUEUED and the data type is PM_Q_STD. SplStdOpen must not be called at any other time.

Parameters

hdc *(HDC)*
Device context handle.

Return Codes

This function returns BOOLEAN (fSuccess):

TRUE Successful.
FALSE Error.

SplStdQueryLength

```
LONG APIENTRY SplStdQueryLength (hstd)
```

This function returns the number of data bytes in the PM_Q_STD buffer identified by hstd.

Parameters

hstd *(HSTD)*

Handle to the PM_Q_STD buffer.

Return Codes

This function returns the size of hstd (cBytes) or, if an error occurred, SPL_ERROR.

The Queue Manager (Print Manager)

Note: The Presentation Manager shell uses the term *Print Manager* as the application name of the spooler queue manager.

The purpose of the queue manager is to control the queues, create new spool files, and to invoke the queue processor when a job is ready for printing. The queue manager also provides a function, SplMessageBox, that can called to display a message to the user.

The following functions are available in the queue manager.

SplQmAbort

SplQmAbortDoc

SplQmClose

SplQmEndDoc

SplQmOpen

SplQmStartDoc

SplQmWrite.

SplMessageBox.

SplQmAbort

```
BOOL   APIENTRY SplQmAbort (hspl)
```

Aborts and closes the spool file identified by hspl.

Parameters

hspl *(HSPL)*
Spooler handle.

Return Codes

This function returns BOOLEAN (fSuccess):

TRUE	Successful.
FALSE	Error.

```
BOOL   APIENTRY SplQmAbortDoc (hspl)
```

Aborts the document on the spool file identified by hspl. All data for that document, including SplQmStartDoc data, is erased.

SplQmAbortDoc does not close the spool file; the presentation driver can restart the document using the same spooler handle. If the driver wants to abort the job and close the file, it should call SplQmAbort.

Parameters

hspl *(HSPL)*
Spooler handle.

Return Codes

This function returns BOOLEAN (fSuccess):

TRUE	Successful.
FALSE	Error.

SplQmClose

```
BOOL   APIENTRY SplQmClose (hspl)
```

Closes the spool file identified by hspl. (SplQmClose corresponds to DevCloseDC.)

Parameters

hspl *(HSPL)*
Spooler handle.

Return Codes

This function returns BOOLEAN (fSuccess):

TRUE	Successful.
FALSE	Error.

```
USHORT APIENTRY SplQmEndDoc (hspl)
```

Ends the document on the spool file identified by hspl. (SplQmEndDoc corresponds to the DEVESC_ENDDOC escape code.)

The return code, if it is not SPL_ERROR, is the queue manager's job ID for the document.

Parameters

hspl *(HSPL)*
Spooler handle.

Return Codes

idJobId Job identifier.
SPL_ERROR Error.

SplQmOpen

```
HSPL  APIENTRY SplQmOpen (pszToken, cbData, pbData)
```

Opens the queue manager for output to a spool file. (This function is similar to SplStdOpen.)

The return code, if it is not SPL_ERROR, is the handle that identifies the spool file.

Parameters

pszToken *(PSZ)*
Pointer to token, this is a dummy parameter and should be specified as '**'.

cbData *(LONG)*
Number of elements in the data structure.

pbData *(PQMOPENDATA)*
Pointer to a DEVOPENSTRUC structure containing information from the presentation driver's physical device block (see "Remarks" on page 11-10):

pszLogAddress Pointer to the name of the queue.

pszDriverName Pointer to the name of the presentation device driver.

pdriv Pointer to a DRIVDATA structure:

- **cb** Number of bytes in structure.
- **lVersion** Version number.
- **szDeviceName[32]** Device name.
- **abGeneralData** Driver-specific data.

pszDataType Pointer to a string defining the data type of the queued file. All queue processors must support PM_Q_STD and PM_Q_RAW.

pszComment Pointer to a natural language description of the file.

pszQueueProcName Pointer to the name of the queue processor.

pszQueueProcParams Pointer to a string of queue processor parameters.

pszSpoolerParams Pointer to a string of spooler parameters separated by one or more blanks. Valid parameters are:

FORM = aaa
Identifies the form name for the print job; multiple names should be separated by commas (aaa,bbb,ccc). If this parameter is not present in the string of spooler parameters, the job is printed on the current form.

Form names are defined by the presentation driver; valid names are those that would be returned from a call to the driver's GreQueryHardcopyCaps handling routine.

PRTY = n
Identifies the priority for the print job. The priority can be any value from 1 through 99 (1 is lowest priority). If this parameter is not present, the priority value defaults to 50.

pszNetworkParams Pointer to string of networking parameters. These are only used in a network environment and their nature is defined by the network application.

Return Codes

This function returns the spooler handle (hspl) or, if an error occurred, SPL_ERROR.

```
BOOL   APIENTRY SplQmStartDoc (hspl, pszDocName)
```

Signals the start of the document for the spool file and supplies a name that the spooler can use to identify the job to the user.

This function corresponds to Escape (DEVESC_STARTDOC).

Parameters

hspl *(HSPL)*
Spooler handle.

pszDocName *(PSZ)*
Document name, this may be displayed by the spooler to the user.

Return Codes

This function returns BOOLEAN (fSuccess):

TRUE Successful.
FALSE Error.

SplQmWrite

```
BOOL   APIENTRY SplQmWrite (hspl, cbData, pbData)
```

Writes data from the presentation driver's buffer to the spool file.

Parameters

hspl *(HSPL)*
Spooler handle.

cbData *(LONG)*
Length of the buffer, in bytes.

pbData *(PBYTE)*
Pointer to the start of the buffer.

Return Codes

This function returns BOOLEAN (fSuccess):

TRUE	Successful.
FALSE	Error.

Remarks

The size of the data buffer must not be greater than 64KB. Print jobs that exceed the maximum buffer size must be written using repeated calls to this function.

```
USHORT  APIENTRY SplMessageBox (pszAddress, fsErrorInfo, fsErrorData, pszText, pszCaption, idWindow, fsStyle)
```

This function creates and displays a message box.

SplMessageBox is a ring 2 function that is similar to WinMessageBox. For details, see WinMessageBox in the *Programming Reference*.

Parameters

pszAddress *(PSZ)*
Pointer to a string containing the logical address of the device, such as 'LPT1'.

fsErrorInfo *(USHORT)*
Error information. One of the following flags must be set to identify where the error occurred:

SPLINFO_QPERROR	Spooler queue processor error.
SPLINFO_DDERROR	Presentation driver error.
SPLINFO_SPLERROR	Spooler error.
SPLINFO_OTHERERROR	Any other error.

One of the following flags is also set to indicate the severity of the error:

SPLINFO_INFORMATION	Information only, no error.
SPLINFO_WARNING	Warning.
SPLINFO_ERROR	Recoverable error.
SPLINFO_SEVERE	Severe, unrecoverable error.
SPLINFO_USERINTREQD	This flag is optional; it shows that recovery requires action from the user.

fsErrorData *(USHORT)*
Error data:

SPLDATA_PRINTERJAM	Printer is jammed, offline, or not powered on.
SPLDATA_FORMCHGREQD	Form change required.
SPLDATA_CARTCHGREQD	Font cartridge change required.
SPLDATA_PENCHGREQD	Pen change required.
SPLDATA_DATAERROR	Data error, such as missing file.
SPLDATA_UNEXPECTERROR	Unexpected DOS error.
SPLDATA_OTHER	Any other error.

pszText *(PSZ)*
Pointer to the text string for the message box.

pszCaption *(PSZ)*
Pointer to a string containing a meaningful title for the message box.

The text is centered in the title bar; if more than 40 characters are supplied, excess characters at the beginning and end of the string are not displayed.

idWindow *(USHORT)*
Window ID of the message box window.

fsStyle *(USHORT)*
This bit array specifies the contents and function of the message box.

Return Codes

This function returns a USHORT value (sResponse) that indicates the user's response.

The Queue Processor (Queue Driver)

Note: The Presentation Manager shell uses the term *Queue Driver* to identify the queue processor.

A queue processor takes a spool file and prints it. Each queue has its own queue processor. The Presentation Manager system queue processor is supplied in the file PMPRINT.QPR. You may supply queue processors of your own to support user data types, but any queue processor that you create must support the PM_Q_STD and PM_Q_RAW standard data types.

How a Queue Processor Prints

The method used by the queue processor to write data to the printer depends on the data type:

PM_Q_STD

Data is written to the printer using the API:

Component	Action
Queue processor	Opens a DC for the printer using DevOpenDC and enables it as an OD_DIRECT device for PM_Q_STD data.
QP	Issues Escape(DEVESC_STARTDOC).
QP	Writes data, which is in metafile format, to DC.
Driver	Starts a journal file.
Driver	Processes and journals each call as it is passed by the queue processor.
QP	Issues Escape(DEVESC_ENDDOC).
Driver	Stops the journal file.
Driver	Plays each band to the device using the Prt... interface.
QP	Closes printer DC using DevCloseDC.
Driver	Closes printer using PrtClose.

PM_Q_RAW

Component	Action
Queue processor	Opens a DC for the printer using DevOpenDC and enables it as an OD_DIRECT device for PM_Q_RAW data.
QP	Issues Escape(DEVESC_STARTDOC).
Driver	Opens the device for printing, using PrtOpen.
QP	Writes the data to the DC.
Driver	Sends the data to the device without further processing. The data is buffered into blocks of convenient size and written to the printer using PrtWrite. Control codes are sent to the printer using the PrtDevIOCtl commands. The size of the block depends on the amount of memory available to the device driver.
QP	Closes printer DC using DevCloseDC.
Driver	Closes printer using PrtClose.

User Data Types

The processing required for user data types depends on the format of the data type. In some instances it may be necessary to create a special queue processor to support the data type.

The following functions provide an interface to the queue processor:

SplQpClose
SplQpControl
SplQpInstall
SplQpOpen
SplQpPrint
SplQpQueryDt.

In addition to these functions, a visual interface is supplied through the SplMessageBox entry point.

```
BOOL APIENTRY SplQpClose (hproc)
```

Closes the queue processor.

Parameters

hproc *(HPROC)*
Queue processor handle.

Return Codes

This function returns BOOLEAN (fSuccess):

TRUE	Successful.
FALSE	Error.

SplQpControl

```
BOOL APIENTRY SplQpControl (hproc, cmdCode)
```

Controls the printing of a document.

Parameters

hproc *(HPROC)*
processor handle.

cmdCode *(LONG)*
Control code:

SPLC_ABORT	Printing is aborted and the queue processor is closed.
SPLC_PAUSE	Printing is paused. The spool file must not be open, or allocated by the queue processor.
SPLC_CONTINUE	Printing resumes on a paused job. The spool file is unaltered.

Return Codes

This function returns BOOLEAN (fSuccess):

TRUE	Successful.
FALSE	Error.

```
BOOL APIENTRY SplQpInstall (hwnd)
```

Allows the user to install a queue processor, using a control panel window.

Parameters

hwnd *(HWND)*
Control panel window handle.

Return Codes

This function returns BOOLEAN (fSuccess):

TRUE	Successful.
FALSE	Error.

Remarks

This function is called by the control panel when the queue processor is installed. The user can then hold a dialog with the processor to obtain processor-dependent values which are then stored in the initialization (.INI) file.

SplQpOpen

```
HPROC APIENTRY SplQpOpen (cbData, pbData)
```

Opens the queue processor and returns its handle. This function is normally called by the spooler-queue manager.

Parameters

cbData *(LONG)*
Number of elements in the SQPOPENDATA structure.

pbData *(PSQPOPENDATA)*
Pointer to SQPOPENDATA structure. This structure is based on the DEVOPENSTRUC and would typically contain data extracted from the presentation driver's physical device block.

pszLogAddress Pointer to logical address.

pszDriverName Pointer to presentation driver name.

pdriv Pointer to a DRIVDATA structure:

- **cb** Size, in bytes, of this structure.
- **lVersion** Version number of the data. Version numbers are defined by the presentation driver.
- **szDeviceName[32]** String identifying the device. Valid values are supplied by the presentation driver.
- **abGeneralData** General data, the type of which is defined by the presentation driver.

pszDataType Data type of the queued file. All queue processors must support PM_Q_STD and PM_Q_RAW; user-defined data types are optional.

pszComment Pointer to a natural language description of the file which could, for example, be displayed by the spooler to the user.

pszProcParams Pointer to a string of queue processor parameters.

pszSpoolerParams Pointer to a string of spooler parameters separated by one or more blanks. Valid parameters are:

FORM = aaa
Identifies the form name for the print job; multiple names should be separated by commas (aaa,bbb,ccc). If this parameter is not present, the job is printed on the current form.

Form names are defined by the presentation driver; valid names are those that would be returned from a call to the driver's GreQueryHardcopyCaps handling routine.

PRTY = n
Identifies the priority for the print job. The priority can be any value from 1 through 99 (1 is lowest priority). But note that this parameter has no significance when passed to SplQpOpen.

pszNetworkParams Pointer to string of networking parameters. These are only used in a network environment and their nature is defined by the network application. Typically, the queue processor ignores this parameter

pszDocName Pointer to a string containing the document name.

pszQueueName Pointer to a string containing the name of the queue from which the job was sent by the spooler queue manager.

pszToken Device information token. This identifies additional device information held in the OS2.INI file. The queue processor ignores this parameter.

idJobId This is a USHORT value that identifies the print job.

Return Codes

hproc

hproc Queue processor handle
SPL_ERROR Error.

SplQpPrint

```
BOOL APIENTRY SplQpPrint (hproc, pszFilename)
```

Processes and prints the spool file. The handling routine in the queue processor opens a DC for an OD_DIRECT device-type; this instance of a DC processes the spooled data and, using the Prt... interface passes the output through the kernel device driver to the physical device.

The queue processor assumes that the device is set at the start a new page and, when the print job ends, issues a form feed to the device.

Parameters

hproc *(HPROC)*
Processor handle.

pszFilename *(PSZ)*
Pointer to name of file containing the data.

Return Codes

This function returns BOOLEAN (fSuccess):

TRUE	Successful.
FALSE	Error.

```
BOOL APIENTRY SplQpQueryDt (pcDatatypes,
paszDatatypes)
```

Returns a list of supported data types.

Parameters

pcDatatypes *(PLONG)*
Pointer to a value indicating the maximum number of data types; on return this parameter is updated to the number of data types returned.

paszDatatypes[pcDatatypes][16] *(PSZ)*
Pointer to an array of data type names returned. Each name is contained in a character string with a maximum length of 16 characters, including the terminal zero.

Return Codes

This function returns BOOLEAN (fSuccess):

TRUE Successful.
FALSE Error.

Remarks

This function should be called once with pcDatatypes set to zero, to determine the number of data types. The array apszDatatypes is not updated in this instance. The application can then allocate storage for the array, and call the function a second time to return a list of supported data types.

Chapter 18. File System Emulation

Presentation drivers for hard-copy devices use an internal interface to communicate with the device's kernel device driver. This interface is needed to ensure that all print jobs are routed through the spooler and the queue manager. The technique used by Presentation Manager is to monitor the data passed to the kernel device driver, trap data that did not come from a presentation driver, and pass the trapped data to the spooler. The monitor is installed automatically by the Presentation Manager for hard-copy devices registered through the control panel.

The internal interface is based on the DOS file system calls DosOpen, DosClose, DosWrite, and so on. Presentation drivers open a device and receive a handle that identifies the device as a file. Subsequent operations, such as writing to the device, are implemented by writing to the returned handle.

The functions that the presentation driver uses are:

PrtAbort

PrtClose

PrtDevIOCtl

PrtOpen

PrtWrite.

PrtAbort

```
VOID APIENTRY PrtAbort (hDevice)
```

Aborts operations to the output device identified by hDevice file handle. Any output data that is held in buffers for the kernel device driver is flushed.

PrtAbort does not close the device; the presentation driver must call PrtClose after aborting operations. If PrtWrite is called to write to a device whose output has been aborted, the call is not honored by the system.

Parameters

hDevice *(HFILE)*

Device handle.

Return Codes

None. PrtAbort is a VOID function.

Remarks

Presentation drivers should not use PrtClose to abort an output operation. The effect of PrtClose is to output any buffered data before closing the device.

```
USHORT APIENTRY PrtClose (hDevice)
```

Closes the output device identified by hDevice.

Parameters

hDevice *(HFILE)*
Device handle.

Return Codes

PrtClose returns the same codes as DosClose:

NO_ERROR
ERROR_FILE_NOT_FOUND
ERROR_ACCESS_DENIED
ERROR_INVALID_HANDLE.

Remarks

If the device handle does not correspond to a device in the range LPT1 – LPT3, this function behaves like the file system call DosClose.

If this function returns an error, the call must be reissued to close the device.

PrtDevIOCtl

```
USHORT APIENTRY PrtDevIOCtl (pData, pParms, sFunction, sCategory, hDevice)
```

Passes device-specific commands to the device.

PrtDevIOCtl is an emulation of DosDevIOCtl, for a full description of the parameters, see DosDevIOCtl in the *Programming Reference* For further information about the IOCtl interface, see the chapter on generic IOCtl commands in *I/O Subsystems and Device Support Volume 1*.

Parameters

pData *(PVOID)*
Pointer to a data packet.

pParms *(PVOID)*
Parameter list.

sFunction *(USHORT)*
Function number.

sCategory *(USHORT)*
Category code.

hDevice *(HFILE)*
Device handle.

Return Codes

PrtDevIOCtl returns the same codes as DosDevIOCtl:

NO_ERROR
ERROR_INVALID_FUNCTION
ERROR_INVALID_HANDLE
ERROR_INVALID_DRIVE
ERROR_GEN_FAILURE
ERROR_INVALID_PARAMETER
ERROR_PROTECTION_VIOLATION
ERROR_INVALID_CATEGORY
ERROR_BAD_DRIVER_LEVEL
ERROR_UNCERTAIN_MEDIA
ERROR_MONITORS_NOT_SUPPORTED.

Remarks

If the device handle does not correspond to a device in the range LPT1 – LPT3, this function behaves like the file system call DosDevIOCtl.

```
USHORT APIENTRY PrtOpen  (pszDeviceName, phDevice, psAction, cFile, sFat, fnOpen, fsMode, lRes)
```

Opens a device file for output and returns its handle in the location addressed by phDevice.

If an attempt is made to open a device file that is already open, an error is returned.

PrtOpen is an emulation of DosOpen; for a full description of the parameters, see DosOpen in the *Programming Reference*.

Parameters

pszDeviceName *(PSZ)*
Pointer to a string identifying the device.

phDevice *(PHFILE)*
Pointer to a location for the returned file handle for the device.

psAction *(PUSHORT)*
Near pointer to location for the returned value that identifies the action taken by the system.

cFile *(ULONG)*
Initial size, in bytes, of the output file.

sFat *(USHORT)*
File attribute.

fnOpen *(USHORT)*
Open function type.

fsMode *(USHORT)*
Open mode of file.

lRes *(ULONG)*
Reserved, must be zero.

Return Codes

PrtOpen returns the same codes as DosOpen:

NO_ERROR
ERROR_FILE_NOT_FOUND
ERROR_PATH_NOT_FOUND
ERROR_TOO_MANY_OPEN_FILES
ERROR_ACCESS_DENIED
ERROR_INVALID_ACCESS
ERROR_NOT_DOS_DISK
ERROR_SHARING_VIOLATION
ERROR_SHARING_BUFFER_EXCEEDED
ERROR_CANNOT_MAKE
ERROR_INVALID_PARAMETER
ERROR_DRIVE_LOCKED
ERROR_OPEN_FAILED
ERROR_DISK_FULL
ERROR_FILENAME_EXCED_RANGE
ERROR_PIPE_BUSY
ERROR_DEVICE_IN_USE.

Remarks

If the device name is not in the range LPT1 – LPT3, this function behaves the same as DosOpen, and any subsequent Prt... calls to this device behave like their DOS equivalents.

PrtWrite

```
USHORT APIENTRY PrtWrite (hDevice, pvoidData, cData, cWritten)
```

Writes data to the device file identified by hDevice.

Parameters

hDevice *(HFILE)*
Device handle.

pvoidData *(PVOID)*
Pointer to data buffer.

cData *(USHORT)*
Length of data, in bytes.

cWritten *(PUSHORT)*
Near pointer to a count of bytes actually written to the device.

Return Codes

PrtWrite returns the same codes as DosWrite:

NO_ERROR
ERROR_ACCESS_DENIED
ERROR_INVALID_HANDLE
ERROR_NOT_DOS_DISK
ERROR_LOCK_VIOLATION
ERROR_BROKEN_PIPE.

Remarks

If the device handle does not correspond to a device in the range LPT1 – LPT3, PrtWrite behaves like the file system call DosWrite.

If this function returns an error, the call must be reissued with cData = cData – cWritten to write any remaining bytes.

Appendix A. Syntax Conventions

The programming statements in this book use IBM C/2 syntax; support for code written in C/2 is provided in header files identified by the .H file extension. Macro Assembler/2 (MASM) support is provided in the include files identified by the .INC extension.

Parameter Names

Parameter names are constructed to show the data type of the parameter its to indicate its use:

- A lowercase prefix, of one or more characters, that indicates the data type.
- An optional qualifier, starting with an uppercase letter.

 Where possible, standard names have been used to describe parameters. Where multiple word qualifiers are used, the order of the words is not significant.

For example:

```
hdc          /* device-context handle */
pszFilename  /* long pointer to a character string */
npszItem     /* near pointer to a character string */
```

The following standard base tags, and their associated type names are defined:

Tag	Data Type	Description
f	BOOL	A flag, or Boolean variable. The qualifier describes the condition associated with the flag when it is TRUE: for example, fSuccess is TRUE if successful, FALSE if not; whereas fError is TRUE if an error occurred, and FALSE if no error occurred. For objects of type BOOL, the value 0 implies FALSE, and any nonzero value implies TRUE.
ch	CHAR	Signed 8-bit quantity; a character.
s	SHORT	Signed 16-bit quantity; a short. This is often used in place of us when it does not matter whether the value is signed or unsigned.
l	LONG	Signed 32-bit quantity; a long. This is often used in place of ul when it does not matter whether the value is signed or unsigned.
uch	UCHAR	Unsigned 8-bit quantity.
us	USHORT	Unsigned 16-bit quantity.
ul	ULONG	Unsigned 32-bit quantity.
b	BYTE	Unsigned 8-bit quantity; a byte. Same as uch.
sz	CHAR[]	Null-terminated string of characters.
fb	UCHAR	Byte of flags: array of flags, packed in a BYTE.
fs	USHORT	Short of flags: array of flags, packed in a USHORT.
fl	ULONG	Long of flags: array of flags, packed in a ULONG. The three preceding types are used when more than one flag is combined into a byte, short, or long. Typically, the values are combined with the OR operator. These values are always unsigned.
r	REAL	Real number, single precision 32 bits.
rd	DOUBLE	Real number, double precision 64 bits.
pfn		Pointer to a function.
x		X coordinate.
y		Y coordinate.
sel	SEL	Segment selector.

The following standard prefixes are also defined:

Prefix	Description
p	32-bit pointer. For example, pch is a pointer to a character. This is a far pointer for an 80286 processor.
np	Near Pointer.
a	Array. For example, ach is an array of characters.
i	An index into an array. For example, an ich is used to index an ach.
c	Count. For example, cch is a count of characters.

Prefix	Description
d	Delta, or difference between instances of a type. For example, dx is the difference between two values of x.
h	Handle. A value that uniquely identifies an object, but cannot directly be used to access it; for example, hps is a PS handle.
mp	Mapping array. This prefix is always followed by two base types, rather than just one, and represents the most general case of an array. Mathematically, an array is simply a function mapping the index to the value stored in the array: mp is an abbreviation of map. In the construct mpab, a is the type of the index, and b is the type of the value stored in the array. In most instances, the only type that is important is the type of the value; the index is usually an integer with no other meaning: the 'a' prefix is used in this instance.
off	Offset. Generally used as an offset within a data structure; the actual address of the element within the data structure is derived by adding an offset to a pointer to the beginning of the data structure. Normally, off is a byte offset. For example: `pfoo = (FOO *)((BYTE *)pfooBase + offfoo).`
id	Identifier. This is generally used for values that identify some object; usually the association of the ID value and the object are established by the programmer. For example, all windows are identified by their window ID, which can be set and queried by the programmer.
cmd	Command. Used for command values, typically as function parameters.

Some types of parameter are used in pairs, in which instance the qualifiers used reflect the relationship between the two variables. For example:

First/Last	First and last elements in a set. These are typically used with indexes or pointers (pchFirst, pchLast). Both values represent valid values (compare with Min/Max below): for all valid values of x, xFirst < = x < = xLast. The use of > with First or < with Last is almost always an off-by-one error. For example, to determine whether an ich is within ichFirst and ichLast: `if (ich >= ichFirst && ich <= ichLast)` `...` `A typical loop:` `for (ich = ichFirst; ich <= ichLast; ich++)` `...`
Min/Max	Similar to First/Last except that Max is not a valid value in the set (Min is a valid value). For all valid values of x in the set, xMin < = x < xMax. The use of > with Min or < = with Max is almost always an off-by-one error. For example, to determine whether an ich is within ichMin and ichMax: `if (ich >= ichMin && ich < ichMax)` `...` `A typical loop:` `for (ich = ichMin; ich < /* or != */ ichMax; ich++)` `...` The current value (Cur) qualifier can be used with Min and Max when Min or Max can change over time; for example, pbStackMaxCur.
Old/New	Old and new. Typically used for values or states when it is necessary to compare the old and new states of the value.
Next/Prev	Next and previous. Typically used in situations where items are being enumerated, such as with linked lists.
Src/Dst	Source and destination. Typically used in transfer operations.
T	A temporary value.
Save	A temporary saved value, typically used when saving and restoring some state.
Cur	Current value.

The base types and their prefixes are defined as follows:

Data Type	Prefix
PSZ	psz
PCH	pch
HAB	hab
HPS	hps
HDC	hdc
HRGN	hrgn
HBITMAP	hbmp
PLONG	pl
POINTL	ptl
POINTL	pt
RECTL	rcl
RECTL	rc
HWND	hwnd
WPOINT	wpt
WRECT	wrc
FIXED	fx.

Parameters for defined structures are the defined parameter names.

```
AREADEFS struct
        {
             defSet
             fFlags
             CodePage
        }AREADEFS
```

System-defined constants and flags are represented as two or more uppercase words or mnemonic abbreviations separated by underscores, for example, SYS_CONSTANT and SYS_FLAG.

Return Values

Function handling routines pass full 32-bit return codes back to the calling function, in MASM the return code is passed in the DX:AX registers.

Register Content Preservation

Registers BX, CX, and ES may be destroyed. All other registers must be preserved.

Handles

All handles and pointers are 32-bit values.

Coordinates

All coordinates are passed as signed 32-bit values unless stated otherwise. World coordinates are restricted to the 28 low-order bits and lie within the range F8000000H through 07FFFFFFH.

Appendix B. Initialization File

This appendix describes the entries for the PM_SPOOLER_DD, PM_SPOOLER_PORT, and PM_SPOOLER_QP application names in the applications section of the system initialization file. These entries are significant only to presentation drivers and queue processors, they should not be used by applications. For details of other entries in the initialization file, refer to the *Programming Reference*. See also the *Programming Guide* for the format of initialization files and the function calls used to access the data.

Note: Values in the initialization file are identified by a key name. Dot (.), comma (,), equals (=), and semicolon (;) are reserved characters and must not be used in names.

PM_SPOOLER_DD

Application 'PM_SPOOLER_DD'

Keyname <DriverName.DeviceName>
where .DeviceName may be null.

Type String

Content/value <Spare>;<DriverOptions>;<PrinterNamesAndOptions>;

- <Spare> must be null.
- <DriverOptions> is a field that, if it is not null, contains driver-specific data that applies to all of the user-defined printer names that use this driver.
- <PrinterNamesAndOptions> is a field that, if it is not null, lists the user-defined printer names that use this driver and the driver-specific data, which can be null, for each name. The format of <PrinterNamesAndOptions> is :

 <Name>=<Data>,<Name>=<Data>, ... <Name>=<Data>

Note: Driver-specific data is written by the presentation driver and contains the abGeneralData generated by the driver's OS2_PM_DRV_DEVMODE routine.

Initial value IBM4201=;;;

PM_SPOOLER_PORT

Printer Ports

Application 'PM_SPOOLER_PORT'

Keyname <PrinterPort>
where <PrinterPort> is 'LPT1', 'LPT2', or 'LPT3'.

Type String

Content/value <RedirectPort>

If <RedirectPort> is not null, it identifies the redirection port (for example, COM1).

Initial values
LPT1=;
LPT2=;
LPT3=;

Note: Redirection, if specified, is handled by the system; presentation drivers do not need to know about redirection.

Communications Ports

Application 'PM_SPOOLER_PORT'

Keyname <COMx>
where <COMx> is 'COM1', 'COM2', or 'COM3'.

Type String

Content/value <Baud>;<Parity>;<WordLength>;<Stop>;<Handshake>;

Null is a valid value for any field. If a field is not null:

- <Baud> defines the Baud rate.
- <Parity> defines the parity:
 '0' None
 '1' Odd
 '2' Even.
- <WordLength> defines the word length ('4', '5', '6', '7', or '8' bits).
- <Stop> defines the number of stop bits ('1', '1.5', or '2').
- <Handshake> defines the handshaking:
 '0' None
 '1' Hardware.

Initial values None

PM_SPOOLER_QP

Application 'PM_SPOOLER_QP'

Keyname <QueueProcessorName>

Type String

Content/value <FullFileName>;<Options>;

- <FullFileName> is the full file name of the Queue Processor.
- <Options>, if it is not null, is the Queue Processor options.

Initial value `PMPRINT=C:\OS2\DLL\PMPRINT.QPR;;`

Glossary

This glossary defines terms used in this book. It contains terms specific to the Presentation Manager, but it is not a complete glossary for OS/2 Version 1.2. This glossary includes terms and definitions from the *IBM Dictionary of Computing*, SC20-1699.

A

accelerator. A single key stroke that invokes an application-defined function.

accelerator table. Used to define which keystrokes are treated as accelerators, and the commands they are translated into.

action. One of a set of defined tasks a computer performs. Users request the application to perform an action in several ways: typing a command, pressing a function key, or selecting the action name from an action bar or menu.

action bar. The area at the top of a panel that contains keywords that give users access to actions available in the current panel. When users select an action bar choice, a group of actions or additional keywords appear in a pull-down extension from the action bar.

action point. The current position on the screen at which the pointer is pointing. (Contrast with *hot spot* and *input focus*.)

active program. A program currently running on the computer. See also *interactive program*, *noninteractive program*, and *foreground program*.

active window. The window with which the user is currently interacting.

alphanumeric video output. Output to the logical video buffer when the video adapter is in text mode and the logical video buffer is addressed by an application as a rectangular array of character cells.

anchor block. An area of Presentation Manager-internal resources allocated to a process or thread that calls WinInitialize.

anchor point. A point in a window used by a program designer or by a window manager to position a subsequently appearing window.

ANSI. American National Standards Institute.

APA. All points addressable.

API. Application programming interface. The formally-defined programming language that is between an IBM application program and the user of the program. See also GPI.

area. In computer graphics, a filled shape such as a solid rectangle.

ASCII. American National Standard Code for Information Interchange.

ASCIIZ. A string of ASCII characters that is terminated with a byte containing the value 0 (null).

aspect ratio. In computer graphics, the width-to-height ratio of an area, symbol, or shape.

asynchronous. (1) Without regular time relationship. (2) Unexpected or unpredictable with respect to the execution of a program's instructions.

atom. A constant that represents a string. Once a string has been defined as an atom, the atom can be used in place of the string to save space. Strings are associated with their respective atoms in an **atom table.** See also **integer atom.**

atom table. Used to relate **atoms** with the strings that they represent. Also in the table is the mechanism by which the presence of a string can be checked.

attributes. Characteristics or properties that can be controlled, usually to obtain a required appearance; for example, the color of a line. See also *graphics attributes* and *segment attributes*.

AVIO. Advanced Video Input/Output.

B

background color. The color in which the background of a graphic primitive is drawn.

background mix. An attribute that determines how the background of a graphic primitive is combined with the existing color of the graphics presentation space. Contrast with *mix*.

Bezier curves. A mathematical technique of specifying smooth continuous lines and surfaces, which require a starting point and a finishing point with several intermediate points that influence or control the path of the linking curve. Named after Dr. P. Bezier.

bit map. A representation in memory of the data displayed on an APA device, usually the screen.

border. A visual indication (for example, a separator line or a background color) of the boundaries of a window.

button. A mechanism on a *pointing device*, such as a mouse, used to request or initiate an action. Contrast with *pushbutton* and *radio button*.

C

cached micro presentation space. A presentation space from a Presentation Manager-owned store of micro presentation spaces. It can be used for drawing to a window only, and must be returned to the store when the task is complete.

cancel. An action that removes the current window or menu without processing it, and returns the previous window.

CASE statement. Provides, in C/2, the body of a window procedure. There is one CASE statement for each message type written to take specific actions.

cell. See *character cell.*

CGA. Color graphics adapter.

chained list. A list in which the data elements may be dispersed but in which each data element contains information for locating the next. Synonym for *linked list.*

character. A letter, digit, or other symbol.

character box. In computer graphics, the boundary that defines, in world coordinates, the horizontal and vertical space occupied by a single character from a character set. See also *character mode.* Contrast with *character cell.*

character cell. The physical, rectangular space in which any single character is displayed on a screen or printer device. Position is addressed by row and column coordinates. Contrast with *character box.*

character code. The means of addressing a character in a character set, sometimes called *code point.*

character mode. The character mode, in conjunction with the font type, determines the extent to which graphics characters are affected by the character box, shear, and angle attributes.

check box. A control window, shaped like a square button on the screen, that can be in a checked or unchecked state. It is used to select one or more items from a list. Contrast with *radio button.*

check mark. The symbol (√) that is used to indicate a selected item on a pull-down.

child window. A window that is positioned relative to another window (either a main window or another child window). Contrast with *parent window.*

choice. An option that can be selected. The choice can be presented as text, as a symbol (number or letter), or as an icon (a pictorial symbol).

class. See *window class.*

class style. The set of properties that apply to every window in a window class.

client area. The area in the center of a window that contains the main information of the window.

clipboard. An area of main storage that can hold data being passed from one Presentation Manager application to another. Various data formats can be stored.

clipping. In computer graphics, removing those parts of a display image that lie outside a given boundary.

clip limits. The area of the paper that can be reached by a printer or plotter.

clipping path. A clipping boundary in world-coordinate space.

code page. An assignment of graphic characters and control-function meanings to all code points.

code point. Synonym for *character code.*

color dithering. A process that simulates a single color on the screen by setting alternate pels, for example, to different colors.

command. (1) The name and parameters associated with an action that can be performed by a program. A command is one form of action request. Users type in the command and enter it. (2) An action users request to interact with the command area.

command area. An area composed of two elements: a command field prompt and a command entry field.

command entry field. An entry field in which users type commands. The entry field is preceded by a command field prompt. These two elements make up the command area.

command line. On a display screen, a display line (usually at the bottom of the screen) in which only commands can be entered.

command prompt. A field prompt showing the location of the command entry field in a panel.

Common Programming Interface. A consistent set of specifications for languages, commands, and calls to enable applications to be developed across all SAA environments. See also *Systems Application Architecture.*

Common User Access. A set of rules that define the way information is presented on the screen, and the techniques for the user to interact with the information.

control. The means by which an operator gives input to an application. A *choice* corresponds to a control.

Control Panel. In the Presentation Manager, a program used to set up user preferences that act globally across the system.

Control Program. The basic function of OS/2 including DOS emulation and the support for keyboard, mouse, and VIO.

control window. A class of window used to handle a specific kind of user interaction. Radio buttons and check boxes are examples.

correlation. The action of determining which element or object within a picture is at a given position on the display. This follows a *pick* operation.

CPI. Common Programming Interface.

current position. The point from which the next primitive will be drawn.

cursor. A symbol displayed on the screen and associated with an input device. The cursor indicates where input from the device will be placed. Types of cursors include text cursors, graphics cursors, and selection cursors. Contrast with *pointer* and *input focus.*

D

data structure. (ISO) The syntactic structure of symbolic expressions and their storage-allocation characteristics.

DBCS. See *double-byte character set.*

default procedure. Function provided by the Presentation Interface that may be used to process standard messages from dialogs or windows.

default value. A value used when no value is explicitly specified by the user. For example, in the graphics programming interface, the default line type is 'solid'.

Desktop Manager. In the Presentation Manager, a window from which users can start one or more listed programs.

desktop window. The window, corresponding to the physical device, against which all other types of windows are established.

device context. A logical description of a data destination such as memory, metafile, display, printer, or plotter. See also *direct device context, information device context, memory device context, metafile device context, queued device context,* and *screen device context.*

device driver. A file that contains the code needed to attach and use a device such as a display, printer, or plotter.

device space. Coordinate space in which graphics are assembled after all GPI transformations have been applied. Device space is defined in device-specific units.

dialog. The interchange of information between a computer and its user through a sequence of requests by the user and the presentation of responses by the computer.

dialog box. A type of window that contains one or more controls for the formatted display and entry of data. Also known as a *pop-up window.* A modal dialog box is used to implement a pop-up window.

Dialog Box Editor. A what-you-see-is-what-you-get (WYSIWYG) editor that creates dialog boxes for communicating with the application user.

dialog item. A component (for example, a menu or a button) of a dialog box. Dialog items are also used when creating dialog templates.

dialog tag language. A markup language used by the DTL compiler to create dialog objects. It is based on the Standard Generalized Markup Language (SGML).

dialog template. The definition of a dialog box, which contains details of its position, appearance, and window ID, and the window ID of each of its child windows.

direct device context. A logical description of a data destination that is a device other than the screen (for example, a printer or plotter), and where the output is not to go through the spooler. Its purpose is to satisfy queries. See also *device context.*

direct manipulation. The action of using the mouse to move objects around the screen. For example, moving files and directories about in the File Manager.

directory. A type of file containing the names and controlling information for other files or other directories.

display point. Synonym for *pel.*

dithering. The process used in color displays whereby every other pel is set to one color, and the intermediate pels are set to another. Together they produce the effect of a third color at normal viewing distances. This process can only be used on solid areas of color; it does not work on narrow lines, for example.

double-byte character set (DBCS). A set of characters in which each character is represented by two bytes. Languages such as Japanese, Chinese, and Korean, which contain more characters than can be represented by 256 code points, require double-byte character sets. As each character requires two bytes, the entering, displaying, and printing of DBCS characters requires hardware and software that can support DBCS.

dragging. In computer graphics, moving an object on the display screen as if it were attached to the pointer.

drawing chain. See *segment chain.*

drop. To fix the position of an object that is being dragged, by releasing the select button of the pointing device.

DTL. See *dialog tag language.*

dynamic segments. Graphics segments drawn in exclusive-OR mix mode so that they can be moved from one screen position to another without affecting the rest of the displayed picture.

E

EBCDIC. Extended binary-coded decimal interchange code.

EGA. Extended graphics adapter.

element. An entry in a graphics segment that comprises one or more graphics orders and that is addressed by the element pointer.

entry field. A panel element in which users type information. Compare to *selection field.*

entry panel. A defined panel type containing one or more entry fields and protected information such as headings, prompts, and explanatory text.

extended help. A facility that provides users with information about an entire application panel rather than a particular item on the panel.

entry-field control. The means by which the application receives data entered by the user in an entry field. When it has the input focus, it displays a flashing pointer at the position where the next typed character will go.

exit. The action that terminates the current function and returns the user to a higher level function. Repeated exit

requests return the user to the point from which all functions provided to the system are accessible. Contrast with *cancel*.

extended-choice selection. A mode that allows the user to select more than one item from a window. Not all windows allow extended choice selection. Contrast with *multiple-choice selection*.

F

field-level help. Information specific to the field on which the cursor is positioned. This help function is "contextual" because it provides information about a specific item as it is currently used; the information is dependent upon the context within the work session.

File Manager. In the Presentation Manager, a program that displays directories and files, and allows various actions on them.

file specification. The full identifier for a file, which includes its file name, extension, path, and drive.

fillet. A curve that is tangential to the end points of two adjoining lines. See also *polyfillet*.

font. A particular size and style of typeface that contains definitions of character sets, marker sets, and pattern sets.

foreground program. The program with which the user is currently interacting. Also known as *interactive program*.

frame. The part of a window that can contain several different visual elements specified by the application, but drawn and controlled by the Presentation Manager. The frame encloses the client area.

frame styles. Different standard window layouts provided by the Presentation Manager.

full-screen application. An application program that occupies the whole screen.

function key. A key that causes a specified sequence of operations to be performed when it is pressed, for example, F1 and Alt-K.

G

Global Descriptor Table (GDT). Defines code and data segments available to all tasks in an application.

glyph. A graphic symbol whose appearance conveys information.

GPI. Graphics Programming Interface. The formally-defined programming language that is between an IBM graphics program and the user of the program. See also *API*.

graphics. A picture defined in terms of graphic primitives and graphics attributes.

graphics attributes. Attributes that apply to graphic primitives. Examples are color, line type, and shading-pattern definition. See also *segment attributes*.

graphics field. The clipping boundary that defines the visible part of the presentation-page contents.

graphics model space. The conceptual coordinate space in which a picture is constructed after any model transforms have been applied. Also known as *model space*.

graphic primitive. A single item of drawn graphics, such as a line, arc, or graphics text string. See also *graphics segment*.

graphics segment. A sequence of related graphic primitives and graphics attributes. See also *graphic primitive*.

graying. The indication that a choice on a pull-down is unavailable.

group. A collection of logically-connected controls. For example, the buttons controlling paper size for a printer. See also *program group*.

H

handle. An identifier that represents an object, such as a device or window, to the Presentation Interface.

hard error. An error condition on a network that requires either that the system be reconfigured, or that the source of the error be removed before the system can resume reliable operation.

heap. An area of free storage available for dynamic allocation by the program. Its size varies, depending on the storage requirements of the program.

help. A function that provides information about a specific field, an application panel, or information about the help facility. It provides field help when the cursor is on a selection or entry field in an application panel or another help panel. It provides information about the application panel, called *extended help*, when the cursor is not in an interactive field.

help index. A facility that allows the user to select topics for which help is available.

help panel. A panel with information to assist users that is displayed in response to a help request from the user.

help window. A Common User Access-defined secondary window that displays information when the user requests help.

hit testing. The means of identifying which window is associated with which input device event.

hook. A mechanism by which procedures are called when certain events occur in the system. For example, the filtering of mouse and keyboard input before it is received by an application program.

hook chain. A sequence of hook procedures that are "chained" together so that each event is passed, in turn, to each procedure in the chain.

hot spot. The part of the pointer that must touch an object before it can be selected. This is usually the tip of the pointer. Contrast with *action point*.

I

icon. A pictorial representation of an item the user can select. Icons can represent items (such as a document file) that the user wants to work on, and actions that the user wants to perform. In the Presentation Manager, icons are used for data objects, system actions, and minimized programs.

icon area. In the Presentation Manager, the area at the bottom of the screen that is normally used to display the icons for minimized programs.

Icon Editor. The Presentation Manager-provided tool for creating icons.

image font. A set of symbols, each of which is described in a rectangular array of pels. Some of the pels in the array are set to produce the image of the symbol. Contrast with *outline font.*

information device context. A logical description of a data destination other than the screen (for example, a printer or plotter), but where no output will occur. Its purpose is to satisfy queries. See also *device context.*

information panel. A defined panel type characterized by a body containing only protected information.

input focus. The area of the screen that will receive input from an input device (typically the keyboard).

input router. OS/2-internal process that removes messages from the system queue.

integer atom. A special kind of **atom** that represents a predefined system constant and carries no storage overhead. For example, names of window classes provided by Presentation Manager are expressed as integer atoms.

interactive graphics. Graphics that can be moved or manipulated by a user at a terminal.

interactive program. A program that is running (active) and is ready to receive (or is receiving) input from the user. Compare with *active program* and contrast with *noninteractive program.*

Also known as a *foreground program.*

interchange file. Data that can be sent from one Presentation Interface application to another.

J

journal. A special-purpose file that is used to record changes made in the system.

K

kanji. A graphic character set used in Japanese ideographic alphabets.

kerning. The design of graphics characters so that their character boxes overlap. Used to space text proportionally.

keys help. A facility that gives users a listing of all the key assignments for the current application.

L

label. In a graphics segment, an identifier of one or more elements that is used when editing the segment.

language support procedure. Function provided by the Presentation Interface for applications that do not, or cannot (as in the case of COBOL and FORTRAN programs), provide their own dialog or window procedures.

LIFO stack. A data stack from which data is retrieved in last-in, first-out order.

linked list. Synonym for *chained list.*

list box. A control window containing a vertical list of selectable descriptions.

list panel. A defined panel type that displays a list of items from which users can select one or more choices and then specify one or more actions to work on those choices.

Local Descriptor Table (LDT). Defines code and data segments specific to a single task.

LVB. Logical Video Buffer.

M

main window. The window that is positioned relative to the desktop window.

marker box. In computer graphics, the boundary that defines, in world coordinates, the horizontal and vertical space occupied by a single marker from a marker set.

marker symbol. A symbol centered on a point. Graphs and charts can use marker symbols to indicate the plotted points.

maximize. A window-sizing action that makes the window the largest size possible.

media window. The part of the physical device (display, printer, or plotter) on which a picture is presented.

memory device context. A logical description of a data destination that is a memory bit map. See also *device context.*

menu. A type of panel that consists of one or more selection fields. Also called a *menu panel.*

message. 1. In Presentation Manager, a packet of data used for communication between the Presentation Interface and windowed applications.

2. In a user interface, information not requested by users but presented to users by the computer in response to a user action or internal process. Messages on status, problems, or user actions from a computer application should be distinguished from a "message" or note sent to users by other users over a communications link.

message filter. The means of selecting which messages from a specific window will be handled by the application.

message queue. A sequenced collection of messages to be read by the application.

metafile. The generic name for the definition of the contents of a picture. Metafiles are used to allow pictures to be used by other applications.

metafile device context. A logical description of a data destination that is a metafile, which is used for graphics interchange. See also *device context*.

metalanguage. A language used to specify another language. In this publication, the data types are described using a metalanguage so as to make the descriptions independent of any one computer language.

mickey. A unit of measurement for physical mouse motion whose value depends on the mouse device driver currently loaded.

micro presentation space. A graphics presentation space in which a restricted set of the GPI function calls is available.

minimize. A window-sizing action that makes the window the smallest size possible. In the Presentation Manager, minimized windows are represented by icons.

mix. An attribute that determines how the foreground of a graphic primitive is combined with the existing color of graphics output. Also known as *foreground mix*. Contrast with *background mix*.

mixed character string. A string containing a mixture of one-byte and *kanji* or *hangeul* (two-byte) characters.

mnemonic. A method of selecting an item on a pull-down by means of typing the highlighted letter in the menu item.

modal dialog box. The type of control that allows the operator to perform input operations on only the current dialog box or one of its child windows. Also known as a *serial dialog box*. Contrast with *parallel dialog box*.

modeless dialog box. The type of control that allows the operator to perform input operations on any of the application's windows. Also known as a *parallel dialog box*. Contrast with *modal dialog box*.

model space. See *graphics model space*.

mouse. A hand-held device that is moved around to position the pointer on the screen.

multiple-choice selection. A mode that allows users to select any number of choices, including none at all. See also *check box*. Contrast with *extended-choice selection*.

multitasking. The concurrent processing of applications or parts of applications. A running application and its data are protected from other concurrently running applications.

N

named pipe. A named object that provides client-to-server, server-to-client, or duplex communication between unrelated processes. Contrast with *unnamed pipe*.

noninteractive program. A program that is running (active) but is not ready to receive input from the user. Compare with *active program*, and contrast with *interactive program*.

nonretained graphics. Graphic primitives that are not remembered by the Presentation Interface once they have been drawn. Contrast with *retained graphics*.

null-terminated string. A string of (n + 1) characters where the (n + 1)th character is the 'null' character (X'00'), and is used to represent an n-character string with implicit length. Also known as 'zero-terminated' string and 'ASCIIZ' string.

O

object window. A window that does not have a parent, but which may have child windows. An object window cannot be presented on a device.

open. To start working with a file, directory, or other object.

outline font. A set of symbols, each of which is created as a series of lines and curves. Contrast with *image font*.

output area. The area of the output device within which the picture is to be displayed, printed, or plotted.

owner window. A window into which specific events that occur in another (owned) window are reported.

P

page viewport. A boundary in device coordinates that defines the area of the output device in which graphics are to be displayed. The presentation-page contents are transformed automatically to the page viewport in device space.

paint. The action of drawing or redrawing the contents of a window.

panel. A particular arrangement of information grouped together for presentation to the user in a window.

panel area. An area within a panel that contains related information. The three major panel areas defined by the Common User Access are the action bar, the function key area, and the panel body.

panel body. The portion of a panel not occupied by the action bar, function key area, title or scroll bars. The panel body may contain protected information, selection fields, and entry fields. The layout and content of the panel body determine the panel type.

panel body area. The part of a window not occupied by the action bar or function key area. The panel body area may contain information, selection fields, and entry fields. Also known as *client area*.

panel body area separator. A line or color boundary that provides users with a visual distinction between two adjacent areas of a panel.

panel definition. A description of the contents and characteristics of a panel. Thus, a panel definition is the application developer's mechanism for predefining the format to be presented to users in a window.

panel ID. A panel element located in the upper left-hand corner of a panel body that identifies that particular panel within the application.

panel title. A panel element that identifies the information in the panel.

paper size. The size of paper, defined in either standard U.S. or European names (for example, A, B, A4), and measured in inches or millimeters respectively.

parallel dialog box. See *modeless dialog box*.

parent window. The window relative to which one or more child windows are positioned. Contrast with *child window*.

pel. The smallest area of a display screen capable of being addressed and switched between visible and invisible states. Synonymous with *display point*, *pixel*, and *picture element*.

pick. To select part of a displayed object using the pointer.

picture chain. See *segment chain*.

picture element. Synonym for *pel*.

pipe. See *named pipe*, *unnamed pipe*.

pixel. Synonym for *pel*.

plotter. An output device that uses pens to draw its output on paper or transparency foils.

pointer. The symbol displayed on the screen that is moved by a pointing device, such as a *mouse*. The pointer is used to point at items that users can select. Contrast with *cursor*.

pointing device. A device (such as a mouse) used to move a pointer on the screen.

pointings. Pairs of x-y coordinates produced by an operator defining positions on a screen with a pointing device, such as a *mouse*.

polyfillet. A curve based on a sequence of lines. It is tangential to the end points of the first and last lines, and tangential also to the midpoints of all other lines. See also *fillet*.

polyline. A sequence of adjoining lines.

pop. To retrieve an item from a last-in-first-out stack of items. Contrast with *push*.

pop-up window. A window that appears on top of another window in a dialog. Each pop-up window must be completed before returning to the underlying window.

Presentation Manager. The OS/2 Control Program plus the visual component that presents, in windows, a graphics-based interface to applications and files installed and running in OS/2.

presentation page. The coordinate space in which a picture is assembled for display.

presentation space (PS). Contains the device-independent definition of a picture.

primary window. The window in which the main dialog between users and the application takes place. In a multi-programming environment, each application starts in its own primary window. The primary window remains for the duration of the application, although the panel displayed will change as the user's dialog moves forward. See also *secondary window*.

primitive. See *graphic primitive*.

primitive attribute. A specifiable characteristic of a graphic primitive. See *graphics attributes*.

print job. The result of sending a document or picture to be printed.

Print Manager. In the Presentation Manager, the part of the spooler that manages the spooling process. It also allows users to view print queues and to manipulate print jobs.

process. An instance of an executing application and the resources it is using.

program details. Information about a program that is specified in the Desktop Manager window and is used when the program is started.

program group. In the Presentation Manager, several programs that can be acted upon as a single entity.

program name. The full file specification of a program. Contrast with *program title*.

program title. The name of a program as it is listed in the Desktop Manager window. Contrast with *program name*.

pull-down. An extension of the *action bar* that displays a list of choices available for a selected action bar choice. After users select an action bar choice, the pull-down appears with the list of choices. Additional *pop-up windows* may appear from pull-down choices to further extend the actions available to users.

push. To add an item to a last-in-first-out stack of items. Contrast with *pop*.

pushbutton. A control window, shaped like a rounded-corner rectangle and containing text, that invokes an immediate action, such as 'enter' or 'cancel'.

Q

queue. A list of print jobs waiting to be printed.

queued device context. A logical description of a data destination (for example, a printer or plotter) where the output is to go through the spooler. See also *device context.*

R

radio button. A control window, shaped like a round button on the screen, that can be in a checked or unchecked state. It is used to select a single item from list. Contrast with *check box.*

reentrant. The attribute of a program or routine that allows the same copy of the program or routine to be used concurrently by two or more tasks.

reference phrase. A word or phrase that is emphasized in a device-dependent manner in order to inform the user that additional information for the word or phrase is available.

reference phrase help. Provides help information on a selectable phrase.

refresh. To update a window, with changed information, to its current status.

region. A clipping boundary in device space.

resource. The means of providing extra information used in the definition of a window. A resource can contain definitions of fonts, templates, accelerators, and mnemonics; the definitions are held in a resource file.

restore. To return a window to its original size or position following a sizing or moving action.

retained graphics. Graphic primitives that are remembered by the Presentation Interface after they have been drawn. Contrast with *nonretained graphics.*

reverse video. A form of alphanumeric highlighting for a character, field, or cursor, in which its color is exchanged with that of its background. For example, changing a red character on a black background to a black character on a red background.

RGB. Red-green-blue. For example "RGB display".

roman. Relating to a type style with upright characters.

S

screen. The physical surface of a workstation or terminal upon which information is presented to users.

screen device context. A logical description of a data destination that is a particular window on the screen. See also *device context.*

scroll bar. A control window, horizontally or vertically aligned, that allows the user to scroll additional data into an associated panel area.

scrollable entry field. An entry field larger than the visible field.

scrollable selection field. A selection field that contains more choices than are visible.

scrolling. Moving a display image vertically or horizontally in a manner such that new data appears at one edge, as existing data disappears at the opposite edge.

secondary window. A type of window associated with the primary window in a dialog. A secondary window begins a secondary and parallel dialog that runs at the same time as the primary dialog.

segment. See *graphics segment.*

segment attributes. Attributes that apply to the segment as an entity, as opposed to the individual primitives within the segment. For example, the visibility or detectability of a segment.

segment chain. All segments in a graphics presentation space that are defined with the 'chained' attribute. Synonymous with *picture chain.*

segment priority. The order in which segments are drawn.

segment store. An area in a normal graphics presentation space where retained graphics segments are stored.

select. To mark or choose an item. Note that *select* means to mark or type in a choice on the screen; *enter* means to send all selected choices to the computer for processing.

select button. The button on a pointing device, such as a mouse, that is pressed to select a menu choice. Also known as button 1.

selection cursor. A type of cursor used to indicate the choice or entry field users want to interact with. It is represented by highlighting the item it is currently positioned on.

selection field. A field containing a list of choices from which the user can select one or more.

semaphore. An object used by multi-threaded applications for signaling purposes and for controlling access to serially reusable resources.

separator. See *panel body area separator.*

serial dialog box. See *modal dialog box.*

serially reusable resource (SRR). A logical resource or object that can be accessed by only one task at a time.

session. A routing mechanism for user interaction via the console.

shadow box. The area on the screen that follows mouse movements and shows what shape the window will take if the mouse button is released.

shear. The tilt of graphics text when each character leans to the left or right while retaining a horizontal baseline.

shutdown. In the Desktop Manager, the procedure required before the computer is switched off to ensure that data is not lost.

sibling windows. Child windows that have the same parent window.

slider box. An area on the scroll bar that indicates the size and position of the visible information in a panel area in relation to the information available. Also known as *thumb mark*.

spline. A sequence of one or more Bezier curves.

spooler. A program that intercepts the data going to printer devices and writes it to disk. The data is printed or plotted when it is complete, and the required device is available. The spooler prevents output from different sources being intermixed.

standard window. A collection of windows that form a panel.

static control. The means by which the application presents descriptive information (for example, headings and descriptors) to the user. The user cannot change this information.

style. See *window style*.

suballocation. The allocation of a part of one extent for occupancy by elements of a component other than the one occupying the remainder of the extent.

switch. An action that moves the input focus from one area to another. This can be within the same window or from one window to another.

switch list. See Task List.

symbolic identifier. A text string that equates to an integer value in an include file, that is used to identify a programming object.

System Menu. In the Presentation Manager, the pull-down in the top left corner of a window that allows it to be moved and sized with the keyboard.

system queue. This is the master queue for all pointer device or keyboard events.

Systems Application Architecture. A formal set of rules that enables applications to be run without modification in different computer environments.

T

tag. A markup language word and its attributes that are entered in the source file to identify parts of a panel or other dialog objects.

Task List. In the Presentation Manager, the list of programs that are active. The list can be used to switch to a program and to stop programs.

template. An ASCII-text definition of an action bar and pull-down menu, held in a resource file, or as a data structure in program memory.

text. Characters or symbols.

text cursor. A symbol displayed in an entry field that indicates where typed input will appear.

text window. Also known as the VIO window. The environment in which OS/2 runs AVIO applications.

thread. A unit of execution within a process.

thumb mark. The portion of the scroll bar that describes the range and properties of the data that is currently visible in a window. Also known as a *slider box*.

tilde. A mark used to denote the character that is to be used as a mnemonic when selecting text items within a menu.

title bar. The area at the top of a window that contains the window title. The title bar is highlighted when that window has the input focus. Contrast with *panel title*.

transform. (1) The action of modifying a picture by scaling, shearing, reflecting, rotating, or translating. (2) The object that performs or defines such a modification; also referred to as a *transformation*.

Tree. In the Presentation Manager, the window in the File Manager that shows the organization of drives and directories.

U

unnamed pipe. A circular buffer, created in memory, used by related processes to communicate with one another. Contrast with *named pipe*.

update region. A system-provided area of dynamic storage containing one or more (not necessarily contiguous) rectangular areas of a window, that are visually invalid or incorrect, and therefore in need of repainting.

User Shell. A component of OS/2 that uses a graphics-based, windowed interface to allow the user to manage applications and files installed and running under OS/2.

V

vector font. A set of symbols, each of which is created as a series of lines and curves. Contrast with *image font* and *outline font*.

VGA. Video graphics array.

viewing pipeline. The series of transformations applied to a graphic object to map the object to the device on which it is to be presented.

viewing window. Clipping boundary that defines the visible part of model space.

VIO. Video Input/Output.

virtual memory (VM). Addressable space that is apparent to the user as the processor storage space, but not having a fixed physical location.

visible region. A window's presentation space, clipped to the boundary of the window and the boundaries of any overlying window.

W

wild-card character. The global file-name characters ? or *.

window. A rectangular area of the screen through which a panel or portion of a panel is displayed. A window can be smaller than or equal in size to the screen. Windows can overlap on the screen and give the appearance of one window being on top of another.

window class. The grouping of windows whose processing needs conform to the services provided by one window procedure.

window coordinates. The means by which a window position or size is defined; measured in device units, or *pels*.

window procedure. Code that is activated in response to a message.

window rectangle. The means by which the size and position of a window is described in relation to the desktop window.

window style. The set of properties that influence how events related to a particular window will be processed.

workstation. A display screen together with attachments such as a keyboard, a local copy device, or a tablet.

world coordinates. Application-convenient coordinates used for drawing graphics.

world-coordinate space. Coordinate space in which graphics are defined before transformations are applied.

WYSIWYG. What You See Is What You Get. A capability that enables text to be displayed on a screen in the same way that it will be formatted on a printer.

Z

z-order. The order in which sibling windows are presented. The topmost sibling window obscures any portion of the siblings that it overlaps; the same effect occurs down through the order of lower sibling windows.

zooming. In graphics applications, the process of increasing or decreasing the size of picture.

Index

A

AbortDoc escape code 12-117
AccumulateBounds 12-84
action bar
 definition of X-1
action, definition of X-1
advanced VIO (AVIO) functions 13-2
Arc 14-5
arc functions
 GreArc 14-5
 GreBoxBoth 14-18
 GreBoxBoundary 14-16
 GreBoxInterior 14-14
 GreFullArcBoth 14-12
 GreFullArcBoundary 14-10
 GreFullArcInterior 14-8
 GreGetArcParameters 14-3
 GreGetDefaultArcParameters 15-31
 GrePartialArc 14-6
 GrePolyFillet 14-20
 GrePolyFilletSharp 14-21
 GrePolyMarker 12-28
 GrePolySpline 14-23
 GreSetDefaultArcParameters 15-32
ARCPARAMS structure 14-4
area attributes 12-54
area functions
 GreAreaSetAttributes 14-25
 GreBeginArea 14-26
 GreBeginPath 14-30
 GreCloseFigure 14-33
 GreDrawBorder 12-43
 GreEndArea 14-28
 GreEndPath 14-32
 GreFillPath 14-34
 GreModifyPath 14-37
 GreOutlinePath 14-36
 GreSelectClipPath 14-41
AREABUNDLE mask 12-54
AREABUNDLE structure 12-54
AREADEFS structure 12-54
attribute functions 12-50
 GreAreaSetAttributes 14-25
 GreDeviceGetAttributes 12-63
 GreDeviceQueryFontAttributes 12-79
 GreDeviceSetAttributes 12-65
 GreDeviceSetGlobalAttribute 12-67
 GreGetDefaultAttributes 15-28
 GreGetStyleRatio 12-77
 GreInitializeAttributes 15-35
 GreQueryFontAttributes 15-53
 GreSetArcParameters 14-4
 GreSetAttributes 15-25
 GreSetDefaultAttributes 15-29
 GreSetGlobalAttribute 15-27
 GreSetStyleRatio 12-78
attributes 12-50
 area 12-54
 character 12-56
 color 12-50
 font 15-47
 image 12-60
attributes *(continued)*
 line 12-52
 marker 12-61
 pattern 12-54
AVIO character cell 13-4
AVIO functions 13-2
 GreCharRect 13-5
 GreCharStr 13-6
 GreDeviceSetAVIOFont 13-9
 GreScrollRect 13-7
 GreUpdateCursor 13-8
 VioGetPSAddress 16-13

B

background mix mode 12-51
banding 10-11
BANDRECT structure 12-126
BBITBLTATTRS structure 12-37
BeginArea 14-26
BeginCloseDC 11-19
BeginPath 14-30
Bezier spline 14-23
bit-map file formats 12-30
bit-map functions
 GreBitblt 12-35
 GreCreateBitmap 15-17
 GreDeleteBitmap 15-19
 GreDeleteSetId 15-37
 GreDeviceCreateBitmap 12-31
 GreDeviceDeleteBitmap 12-33
 GreDeviceSelectBitmap 12-34
 GreDrawBorder 12-43
 GreGetAttributes 15-24
 GreGetBitmapBits 12-46
 GreGetBitmapDimension 15-22
 GreGetBitmapParameters 15-21
 GreGetPel 12-40
 GreImageData 12-42
 GreQueryBitmapHandle 15-41
 GreQueryNumberSetIds 15-38
 GreQuerySetIds 15-39
 GreRegionSelectBitmap 14-75
 GreRestoreScreenBits 13-12
 GreSaveScreenBits 13-11
 GreSelectBitmap 15-20
 GreSetBitmapBits 12-48
 GreSetBitmapDimension 15-23
 GreSetBitmapID 15-42
 GreSetPel 12-41
bit-map simulation 10-10
Bitblt 12-35, 12-45
 GrePaintRegion 14-59
 scrolling 13-7
BITMAPINFO structure 12-46
border functions
 GreDrawBorder 12-43
boundary data
 GPI_BOUNDS 10-6
boundary functions
 GreAccumulateBounds 12-84
 GreBoxBoundary 14-16

boundary functions *(continued)*
GreFullArcBoundary 14-10
GreResetBounds 12-82
boundary, noninclusive 12-36
box functions
GreBoxBoth 14-18
GreBoxBoundary 14-16
GreBoxInterior 14-14
GreGetClipBox 14-61
GreGetRegionBox 14-46
GreQueryTextBox 12-22
BoxBoth 14-18
BoxBoundary 14-16
BoxInterior 14-14
BreakExtra escape code 12-118
bundle attribute functions
bundles 12-50
area 12-54
character 12-56
image 12-60, 12-61
line 12-52
pattern 12-54

C

callback 16-13
calling conventions 10-5
calling ring-3 functions 16-13
character attributes 12-56
character cell 13-4
character mode 12-56
character-set functions
GreDeleteSetId 15-37
GreQueryNumberSetIds 15-38
GreQuerySetIds 15-39
CHARBUNDLE mask 12-56
CHARBUNDLE structure 12-56
CHARDEFS structure 12-56
CharExtra escape code 12-119
CharRect 13-5
CharStr 13-6
CharString 12-17
CharStringPos 12-19
CHDIRN_LEFTRIGHT 12-58
CLI 10-8
clip path 14-24
clipping functions
GreCopyClipRegion 14-76
GreExcludeClipRectangle 14-70
GreGetClipBox 14-61
GreGetClipRects 14-62
GreGetDefaultViewingLimits 15-33
GreGetGlobalViewingXform 14-89
GreGetGraphicsField 14-95
GreGetModelXform 14-85
GreGetPageUnits 14-82
GreGetPageViewport 14-93
GreGetViewingLimits 14-97
GreGetWindowViewportXform 14-87
GreIntersectClipRectangle 14-69
GreNotifyClipChange 12-69
GreOffsetClipRegion 14-64
GrePtVisible 14-65
GreQueryClipRegion 14-67
GreRegionSelectBitmap 14-75
GreSelectClipPath 14-41
GreSelectClipRegion 14-68
clipping functions *(continued)*
GreSelectPathRegion 14-74
GreSetDefaultViewingLimits 15-34
GreSetGraphicsField 14-96
GreSetPageUnits 14-83
GreSetPageViewport 14-94
GreSetupDC 14-77
GreSetViewingLimits 14-98
GreSetXFormRect 14-71
CloseDC 15-6
CloseFigure 14-33
CM_MODE1 12-57
code page
GreGetCodePage 12-85
GreQueryCodePageVector 15-54
GreSetCodePage 12-86
WinQueryProcessCP 11-12
color dithering 12-96, 12-101
color table functions
GreCreateLogColorTable 12-96
GreQueryColorData 12-94
GreQueryColorIndex 12-102
GreQueryLogColorTable 12-95
GreQueryNearestColor 12-101
GreQueryRealColors 12-100
GreQueryRGBColor 12-103
GreRealizeColorTable 12-98
GreUnrealizeColorTable 12-99
CombineRectRegion 14-56
CombineRegion 14-55
CombineShortLineRegion 14-57
command area, definition of X-2
command entry field, definition of X-2
command flags 10-6
COM_ALT_BOUND 10-6
COM_AREA 10-6
COM_BOUND 10-6
COM_CORR 10-6
COM_DEVICE 10-7
COM_DRAW 10-6
COM_PATH 10-7
COM_TRANSFORM 10-7
command prompt, definition of X-2
command, definition of X-2
CompleteOpenDC 11-18
COM_ALT_BOUND flag 10-6
COM_AREA flag 10-6
COM_BOUND flag 10-6
COM_BOUND 10-6
COM_CORR flag 10-6
COM_DEVICE flag 10-7
COM_DRAW flag 10-6
COM_PATH flag 10-7
COM_TRANSFORM flag 10-7
control functions
GreDeath 13-19
GreDeviceSetCursor 13-15
GreErasePS 12-75
GreGetPickWindow 13-17
GreLockDevice 12-87
GreResurrection 13-20
GreSetColorCursor 13-16
GreSetPickWindow 13-18
GreUnlockDevice 12-88
SplQpControl 17-22
conventions
parameter names A-1

Convert 14-99
ConvertWithMatrix 14-103
coordinate values 10-7
CopyClipRegion 14-76
CopyDCLoadData 15-43
CreateBitmap 12-31, 15-17
CreateJournalFile 15-57
CreateLogColorTable 12-96
CreateLogicalFont 15-47
CreateRectRegion 14-52
current position 12-5, 12-6
 GreGetCurrentPosition 12-5
 GreSetCurrentPosition 12-6
cursor 11-20
cursor functions
 GreDeviceSetCursor 13-15
 GreSetColorCursor 13-16
 GreUpdateCursor 13-8
curve styling 12-2

D

DAREABUNDLE structure 12-54
data type
 PM_Q_RAW 10-5
 PM_Q_STD 10-5
DC region validation 16-14
DC type
 OD_DIRECT 10-5
 OD_INFO 10-5
 OD_MEMORY 10-5
 OD_QUEUED 10-5
DCHARBUNDLE structure 12-56
DCI 10-11
Death 13-19
DeleteBitmap 12-33, 15-19
DeleteJournalFile 15-58
DELETERETURN structure 12-33
DeleteSetId 15-37
DestroyRegion 14-53
DEVESC_ABORTDOC 12-117
DEVESC_BREAK_EXTRA 12-118
DEVESC_CHAR_EXTRA 12-119
DEVESC_DRAFTMODE 12-120
DEVESC_ENDDOC 12-121
DEVESC_FLUSHOUTPUT 12-122
DEVESC_GETCP 12-123
DEVESC_GETSCALINGFACTOR 12-124
DEVESC_NEWFRAME 12-125
DEVESC_NEXTBAND 12-126
DEVESC_QUERYESCSUPPORT 12-127
DEVESC_QUERYVIOCELLSIZES 12-128
DEVESC_RAWDATA 12-129
DEVESC_STARTDOC 12-130
DEVESC_STD_JOURNAL 12-131
device capabilities 12-113
 GreQueryDeviceBitmaps 12-105
 GreQueryDeviceCaps 12-106
 GreQueryDevResource 12-108
 GreQueryHardcopyCaps 12-113
device context 10-4
 BeginCloseDC 11-19
 CompleteOpenDC 11-18
 DisableDeviceContext 11-14
 dispatch table 10-3
 EnableDeviceContext 11-12
 instance data 10-5, 11-12
device context *(continued)*
 program stack 10-5
 queued devices 17-1
 ResetDCState 11-17
 RestoreDCState 11-16
 SaveDCState 11-15
device context functions
 CloseDC 15-6
 GetHandle 15-9
 GetProcessControl 15-11
 GreCopyDCLoadData 15-43
 GreDeviceInvalidateVisRegion 13-14
 GreDeviceSetDCOrigin 13-22
 GreGetDCOrigin 13-21
 GreLockDevice 12-87
 GreOpenDC 15-3
 GreQueryEngineVersion 15-15
 GreRestorePath 14-44
 GreRestoreRegion 14-73
 GreRestoreXform 14-101
 GreSavePath 14-43
 GreSaveRegion 14-72
 GreSaveXform 14-100
 GreSetupDC 14-77
 GreUnlockDevice 12-88
 ResetDC 15-7
 RestoreDC 15-14
 SaveDC 15-13
 SetHandle 15-10
 SetProcessControl 15-12
device type
 OD_DIRECT 10-5
 OD_INFO 10-5
 OD_MEMORY 10-5
 OD_QUEUED 10-5
DeviceCreateBitmap 12-31
DeviceDeleteBitmap 12-33
DeviceGetAttributes 12-63
DeviceInvalidateVisRegion 13-14
DeviceModes 11-2
DeviceNames 11-4
DeviceQueryFontAttributes 12-79
DeviceQueryFonts 12-80
DeviceSelectBitmap 12-34
DeviceSetAttributes 12-65
DeviceSetAVIOFont 13-9
DeviceSetCursor 13-15
DeviceSetDCOrigin 13-22
DeviceSetGlobalAttribute 12-67
DEVOPENSTRUC structure 11-9
dialog tag language
 definition of X-3
DIMAGEBUNDLE structure 12-60
dirty visible region 10-8
DisableDeviceContext 11-14
DisablePhysicalDeviceBlock 11-11
dispatch table 10-3
DISPLAYINFO structure 12-108
dithering 12-96
dithering, definition of X-3
DLINEBUNDLE structure 12-52
DMARKERBUNDLE structure 12-61
DOS mode 13-19, 13-20
DosCallBack 16-13
DraftMode escape code 12-120
DrawBorder 12-43
drawing functions

drawing functions *(continued)*
GreGetBoundsData 12-83
DrawLinesInPath 12-10
DRIVDATA structure 11-2
driver name 11-9

E

Enable entry point 11-5
BeginCloseDC 11-19
CompleteOpenDC 11-18
DisableDeviceContext 11-14
DisablePhysicalDeviceBlock 11-11
EnableDeviceContext 11-12
FillLogicalDeviceBlock 11-7
FillPhysicalDeviceBlock 11-9
ResetDCState 11-17
RestoreDCState 11-16
SaveDCState 11-15
EnableDeviceContext 11-12
EndArea 14-28
EndDoc escape code 12-121
EndPath 14-32
entry field, definition of X-3
entry panel
definition of X-3
entry points 10-3
entry points, graphics engine
GetDriverInfo 16-7
InnerGreEntry 16-8
SetDriverInfo 16-9
entry points, presentation driver
graphics engine 10-3
imported 10-4
OS2_PM_DRV_DEVICENAMES 11-4
OS2_PM_DRV_DEVMODE 11-2
OS2_PM_DRV_ENABLE 11-5
entry points, system
EqualRegion 14-58
ErasePS 12-75
error codes 10-13
general 10-13
error logging
severity levels 10-13
WinSetErrorInfo 16-15
error severity levels 10-13
error strategy 10-12
ERRORID structure 16-15
Escape 12-115
escape code 12-115
DEVESC_ABORTDOC 12-117
DEVESC_BREAK_EXTRA 12-118
DEVESC_CHAR_EXTRA 12-119
DEVESC_DRAFTMODE 12-120
DEVESC_ENDDOC 12-121
DEVESC_FLUSHOUTPUT 12-122
DEVESC_GETCP 12-123
DEVESC_GETSCALINGFACTOR 12-124
DEVESC_NEWFRAME 12-125
DEVESC_NEXTBAND 12-126
DEVESC_QUERYESCSUPPORT 12-127
DEVESC_QUERYVIOCELLSIZES 12-128
DEVESC_RAWDATA 12-129
DEVESC_STARTDOC 12-130
DEVESC_STD_JOURNAL 12-131
GreEscape 12-115
ExcludeClipRectangle 14-70
exported entry points 10-3
extended help
definition of X-3

F

fast-safe RAM semaphores
FSRSemCheck 16-2
FSRSemEnter 16-3
FSRSemExit 16-5
FSRSemLeave 16-6
FATTRS structure 15-47
field-level help
definition of X-4
figure functions
GreCloseFigure 14-33
file system emulation 18-1
PrtAbort 18-2
PrtClose 18-3
PrtDevIOCtl 18-4
PrtOpen 18-5
PrtWrite 18-6
fillet 12-3
fillet functions
GrePolyFillet 14-20
GrePolyFilletSharp 14-21
FillLogicalDeviceBlock 11-7
FillPath 14-34
FillPhysicalDeviceBlock 11-9
flags 10-6
COM_ALT_BOUND 10-6
COM_AREA 10-6
COM_CORR 10-6
COM_DEVICE 10-7
COM_DRAW 10-6
COM_PATH 10-7
COM_TRANSFORM 10-7
FlushOutput escape code 12-122
font
attributes 15-47
physical 15-45
private 15-45
selection 15-45
font functions
GreCreateLogicalFont 15-47
GreDeviceQueryFontAttributes 12-79
GreDeviceQueryFonts 12-80
GreDeviceSetAVIOFont 13-9
GreGetCodePage 12-85
GreGetPairKerningTable 12-64
GreLoadFont 15-50
GreQueryCodePageVector 15-54
GreQueryFontAttributes 15-53
GreQueryFontFileDescriptions 15-55
GreQueryFonts 15-52
GreQueryLogicalFont 15-46
GreQueryWidthTable 12-26
GreRealizeFont 12-72
GreUnLoadFont 15-51
format, bit-map file 12-30
FSRSEM structure 16-3
FSRSemCheck 16-2
FSRSemEnter 16-3
FSRSemExit 16-5
FSRSemLEAVE 16-6
FullArcBoth 14-12
FullArcBoundary 14-10

FullArcInterior 14-8
function key, definition of X-4
function number 10-6
functions, presentation driver interface 10-3

G

geometric wide line 14-39
GetArcParameters 14-3
GetAttributes 15-24
GetBitmapBits 12-46
GetBitmapDimension 15-22
GetBitmapParameters 15-21
GetClipBox 14-61
GetClipRects 14-62
GetCodePage 12-85
GetCodePage escape code 12-123
GetCurrentPosition 12-5
GetDCOrigin 13-21
GetDefaultArcParameters 15-31
GetDefaultAttributes 15-28
GetDefaultViewingLimits 15-33
GetDriverInfo 16-7
GetGlobalViewingXform 14-89
GetGraphicsField 14-95
GetHandle 15-9
GetLineOrigin 12-89
GetModelXform 14-85
GetPageUnits 14-82
GetPageViewport 14-93
GetPel 12-40
GetPickWindow 13-17
GetProcessControl 15-11
GetRegionBox 14-46
GetRegionRects 14-47
GetScalingFactor escape code 12-124
GetViewingLimits 14-97
GetWindowViewportXform 14-87
GPI_BOUNDS 10-6
graphics engine entry points
 GetDriverInfo 16-7
 SetDriverInfo 16-9
graphics engine, entry points 10-3
GreAccumulateBounds 12-84
GreArc 14-5
GreAreaSetAttributes 14-25
GreBeginArea 14-26
GreBeginPath 14-30
GreBitblt 12-35
 scrolling 13-7
GreBoxBoth 14-18
GreBoxBoundary 14-16
GreBoxInterior 14-14
GreCharRect 13-5
GreCharStr 13-6
GreCharString 12-17
GreCharStringPos 12-19
GreCloseDC 15-6
GreCloseFigure 14-33
GreCombineRectRegion 14-56
GreCombineRegion 14-55
GreCombineShortLineRegion 14-57
GreConvert 14-99
GreConvertWithMatrix 14-103
GreCopyClipRegion 14-76
GreCopyDCLoadData 15-43
GreCreateBitmap 15-17
GreCreateJournalFile 15-57
GreCreateLogColorTable 12-96
GreCreateLogicalFont 15-47
GreCreateRectRegion 14-52
GreDeath 13-19
GreDeleteBitmap 15-19
GreDeleteJournalFile 15-58
GreDeleteSetId 15-37
GreDestroyRegion 14-53
GreDeviceCreateBitmap 12-31
GreDeviceDeleteBitmap 12-33
GreDeviceGetAttributes 12-63
GreDeviceInvalidateVisRegion 13-14
GreDeviceQueryFontAttributes 12-79
GreDeviceQueryFonts 12-80
GreDeviceSelectBitmap 12-34
GreDeviceSetAttributes 12-65
GreDeviceSetAVIOFont 13-9
GreDeviceSetCursor 13-15
GreDeviceSetDCOrigin 13-22
GreDeviceSetGlobalAttribute 12-67
GreDrawBorder 12-43
GreDrawLinesInPath 12-10
GreEndArea 14-28
GreEndPath 14-32
GreEqualRegion 14-58
GreErasePS 12-75
GreEscape 12-115
 DEVESC_ABORTDOC 12-117
 DEVESC_BREAK_EXTRA 12-118
 DEVESC_CHAR_EXTRA 12-119
 DEVESC_DRAFTMODE 12-120
 DEVESC_ENDDOC 12-121
 DEVESC_FLUSHOUTPUT 12-122
 DEVESC_GETCP 12-123
 DEVESC_GETSCALINGFACTOR 12-124
 DEVESC_NEWFRAME 12-125
 DEVESC_NEXTBAND 12-126
 DEVESC_QUERYESCSUPPORT 12-127
 DEVESC_QUERYVIOCELLSIZES 12-128
 DEVESC_RAWDATA 12-129
 DEVESC_STARTDOC 12-130
 DEVESC_STD_JOURNAL 12-131
GreExcludeClipRectangle 14-70
GreFillPath 14-34
GreFullArcBoth 14-12
GreFullArcBoundary 14-10
GreFullArcInterior 14-8
GreGetArcParameters 14-3
GreGetAttributes 15-24
GreGetBitmapBits 12-46
GreGetBitmapDimension 15-22
GreGetBitmapParameters 15-21
GreGetBoundsData 12-83
GreGetClipBox 14-61
GreGetClipRects 14-62
GreGetCodePage 12-85
GreGetCurrentPosition 12-5
GreGetDCOrigin 13-21
GreGetDefaultArcParameters 15-31
GreGetDefaultAttributes 15-28
GreGetDefaultViewingLimits 15-33
GreGetGlobalViewingXform 14-89
GreGetGraphicsField 14-95
GreGetHandle 15-9
GreGetLineOrigin 12-89
GreGetModelXform 14-85

GreGetPageUnits 14-82
GreGetPageViewport 14-93
GreGetPairKerningTable 12-64
GreGetPel 12-40
GreGetPickWindow 13-17
GreGetProcessControl 15-11
GreGetRegionBox 14-46
GreGetRegionRects 14-47
GreGetStyleRatio 12-77
GreGetViewingLimits 14-97
GreGetWindowViewportXform 14-87
GreImageData 12-42
GreInitializeAttributes 15-35
GreIntersectClipRectangle 14-69
GreLoadFont 15-50
GreLockDevice 12-87
GreModifyPath 14-37
GreMultiplyXforms 14-102
GreNotifyClipChange 12-69
GreNotifyTransformChange 12-70
GreOffsetClipRegion 14-64
GreOffsetRegion 14-49
GreOpenDC 15-3
GreOpenJournalFile 15-65
GreOutlinePath 14-36
GrePaintRegion 14-59
GrePartialArc 14-6
GrePlayJournalFile 15-63
GrePolyFillet 14-20
GrePolyFilletSharp 14-21
GrePolyLine 12-8
GrePolyMarker 12-28
GrePolyScanLine 12-14
GrePolyShortLine 12-12
GrePolySpline 14-23
GrePtInRegion 14-50
GrePtVisible 14-65
GreQueryBitmapHandle 15-41
GreQueryCharPositions 12-24
GreQueryClipRegion 14-67
GreQueryCodePageVector 15-54
GreQueryColorData 12-94
GreQueryColorIndex 12-102
GreQueryDeviceBitmaps 12-105
GreQueryDeviceCaps 12-106
GreQueryDevResource 12-108
GreQueryEngineVersion 15-15
GreQueryFontAttributes 15-53
GreQueryFontFileDescriptions 15-55
GreQueryFonts 15-52
GreQueryHardcopyCaps 12-113
GreQueryLogColorTable 12-95
GreQueryLogicalFont 15-46
GreQueryNearestColor 12-101
GreQueryNumberSetIds 15-38
GreQueryRealColors 12-100
GreQueryRGBColor 12-103
GreQuerySetIds 15-39
GreQueryTextBox 12-22
GreQueryWidthTable 12-26
GreRealizeColorTable 12-98
GreRealizeFont 12-72
GreRectInRegion 14-51
GreRectVisible 14-66
GreRegionSelectBitmap 14-75
GreResetBounds 12-82
GreResetDC 15-7
GreRestoreDC 15-14
GreRestorePath 14-44
GreRestoreRegion 14-73
GreRestoreScreenBits 13-12
GreRestoreXform 14-101
GreRestoreXformData 14-92
GreResurrection 13-20
GreSaveDC 15-13
GreSavePath 14-43
GreSaveRegion 14-72
GreSaveScreenBits 13-11
GreSaveXform 14-100
GreSaveXformData 14-91
GreScrollRect 13-7
GreSelectBitmap 15-20
GreSelectClipPath 14-41
GreSelectClipRegion 14-68
GreSelectPathRegion 14-74
GreSetArcParameters 14-4
GreSetAttributes 15-25
GreSetBitmapBits 12-48
GreSetBitmapDimension 15-23
GreSetBitmapID 15-42
GreSetCodePage 12-86
GreSetColorCursor 13-16
GreSetCurrentPosition 12-6
GreSetDefaultArcParameters 15-32
GreSetDefaultAttributes 15-29
GreSetDefaultLimits 15-34
GreSetGlobalAttribute 15-27
GreSetGlobalViewingXform 14-90
GreSetGraphicsField 14-96
GreSetHandle 15-10
GreSetLineOrigin 12-90
GreSetModelXform 14-86
GreSetPageUnits 14-83
GreSetPageViewport 14-94
GreSetPel 12-41
GreSetPickWindow 13-18
GreSetProcessControl 15-12
GreSetRectRegion 14-54
GreSetStyleRatio 12-78
GreSetupDC 14-77
GreSetViewingLimits 14-98
GreSetWindowViewportXform 14-88
GreSetXFormRect 14-71
GreStartJournalFile 15-60
GreStopJournalFile 15-62
GreStrokePath 14-39
GreUnLoadFont 15-51
GreUnlockDevice 12-88
GreUnrealizeColorTable 12-99
GreUpdateCursor 13-8

H

HCINFO structure. 12-113
HDC_IS_DIRTY 10-8
hDDC 11-12
help
 definition of X-4
help index
 definition of X-4
help panel
 definition of X-4
help window
 definition of X-4

I

image attributes 12-60
IMAGEBUNDLE mask 12-60
IMAGEBUNDLE structure 12-60
ImageData 12-42
imported entry points 10-4
include files
 OS2.H 10-3
 OS2.INC 10-3
 PMDDI.H 10-3
 PMDDI.INC 10-3
information panel
 definition of X-5
initialization file B-1
 PM_SPOOLER_DD B-1
 PM_SPOOLER_PORT B-1
 PM_SPOOLER_QP B-2
InitializeAttributes 15-35
InnerGreEntry 16-8
instance data 10-5, 11-12
 GetDriverInfo 16-7
 SetDriverInfo 16-9
interrupts 10-8
IntersectClipRectangle 14-69
InvalidateVisRegion 13-14
IOCtl
 PrtDevIOCtl 18-4

J

JournalData escape code 12-131
journaling functions
 GreCreateJournalFile 15-57
 GreDeleteJournalFile 15-58
 GreEscape, DEVESC_STD_JOURNAL 12-131
 GreOpenJournalFile 15-65
 GrePlayJournalFile 15-63
 GreStartJournalFile 15-60
 GreStopJournalFile 15-62

K

keys help
 definition of X-5

L

line attributes 12-52, 14-37
line functions
 GreCombineShortLineRegion 14-57
 GreDrawLinesInPath 12-10
 GreGetCurrentPosition 12-5
 GreGetLineOrigin 12-89
 GrePolyLine 12-8
 GrePolyScanLine 12-14
 GrePolyShortLine 12-12
 GreSetCurrentPosition 12-6
 GreSetLineOrigin 12-90
LINE structure 12-10
line styling 12-2
LINEBUNDLE mask 12-52
LINEBUNDLE structure 12-52
LINEDEFS structure 12-52
list panel
 definition of X-5
LoadFont 15-50
LockDevice 12-87
logical address 11-9
logical color table 12-92
logical color table functions
 GreCreateLogColorTable 12-96
 GreQueryColorData 12-94
 GreQueryColorIndex 12-102
 GreQueryLogColorTable 12-95
 GreQueryNearestColor 12-101
 GreQueryRealColors 12-100
 GreQueryRGBColor 12-103
 GreRealizeColorTable 12-98
 GreUnrealizeColorTable 12-99
logical device block 11-7
logical video buffer 13-2
LVB 13-2

M

magic cookie 10-11
mandatory functions
 all devices 12-1
 display devices 13-1
marker attributes 12-61
marker functions
 GrePolyMarker 12-28
MARKERBUNDLE mask 12-61
MARKERBUNDLE structure 12-61
MARKERDEFS structure 12-61
mask
 AREABUNDLE 12-54
 CHARBUNDLE 12-56
 IMAGEBUNDLE 12-60
 LINEBUNDLE 12-52
 MARKERBUNDLE 12-61
memory management
 SSAllocHuge 16-10
 SSAllocSeg 16-11
 SSFreeSeg 16-12
message
 definition of X-5
message, user
 SplMessageBox 17-19
mix modes 12-50
ModifyPath 14-37
module definition file 10-3
monochrome device 12-93
MoveCursor 11-20
MultiplyXforms 14-102

N

naming conventions A-1
NewFrame escape code 12-125
NextBand escape code 12-126
noninclusive boundary 12-36
NotifyClipChange 12-69
NotifyTransformChange 12-70
NOTIFYTRANSFORMDATA structure 12-71

O

OD_DIRECT 10-5
OD_INFO 10-5
OD_MEMORY 10-5

OD_QUEUED 10-5
OffsetClipRegion 14-64
OffsetRegion 14-49
OpenDC 15-3
OpenJournalFile 15-65
OS2SYS.INI B-1
OS2.H 10-3
OS2.INC 10-3
OS2_PM_DRV_DEVICENAMES 11-4
OS2_PM_DRV_DEVMODE 11-2
OS2_PM_DRV_ENABLE 11-5
 BeginCloseDC 11-19
 CompleteOpenDC 11-18
 DisableDeviceContext 11-14
 EnableDeviceContext 11-12
 FillLogicalDeviceBlock 11-7
 FillPhysicalDeviceBlock 11-9
 ResetDCState 11-17
 RestoreDCState 11-16
 SaveDCState 11-15
OS/2 mode 13-19, 13-20
OutlinePath 14-36
output banding 10-11

P

PaintRegion 14-59
panel area, definition of X-6
panel body, definition of X-6
panel definition
 definition of X-7
panel ID, definition of X-7
parameter names A-1
PartialArc 14-6
path functions 14-24
 GreBeginPath 14-30
 GreCloseFigure 14-33
 GreDrawLinesInPath 12-10
 GreEndPath 14-32
 GreFillPath 14-34
 GreModifyPath 14-37
 GreOutlinePath 14-36
 GreRestorePath 14-44
 GreSavePath 14-43
 GreSelectClipPath 14-41
 GreSelectPathRegion 14-74
 GreStrokePath 14-39
pattern attributes 12-54
pDCI 10-11, 11-10
physical device block
 disable 11-11
 fill 11-9
physical font 15-45
pInstance 10-5, 11-12
PlayJournalFile 15-63
PMDDI.H 10-3
PMDDI.INC 10-3
PM_Q_RAW 10-5
PM_Q_RAW data
 format 17-2
PM_Q_STD 10-5
PM_Q_STD data
 format 17-1
PM_SPOOLER_DD B-1
PM_SPOOLER_PORT B-1
PM_SPOOLER_QP B-2
POINTL structure 14-5
POINTS structure 13-15
PolyFillet 14-20
PolyFilletSharp 14-21
PolyLine 12-8
PolyMarker 12-28
PolyScanLine 12-14
PolyShortLine 12-12
PolySpline 14-23
PostDeviceModes 11-2
precision 12-56
presentation driver interface 10-3
presentation driver, sample code 10-7
primary window
 definition of X-7
Print Manager i
private font 15-45
process control flags
 GreGetProcessControl 15-11
 GreSetProcessControl 15-12
program stack 10-5
 frame 10-5
PrtAbort 18-2
PrtClose 18-3
PrtDevIOCtl 18-4
PrtOpen 18-5
PrtWrite 18-6
PtInRegion 14-50
PtVisible 14-65
pull-down, definition of X-7

Q

QmAbort 17-12
QmAbortDoc 17-13
QmClose 17-14
QmEndDoc 17-15
QmOpen 17-16
QmStartDoc 17-17
QmWrite 17-18
QpClose 17-21
QpControl 17-22
QpInstall 17-23
QpOpen 17-24
QpPrint 17-26
QpQueryDt 17-27
query functions 12-104
 GreDeviceQueryFontAttributes 12-79
 GreDeviceQueryFonts 12-80
 GreEscape, DEVESC_QUERYESCSUPPORT 12-127
 GreEscape, DEVESC_QUERYVIOCELLSIZES 12-128
 GreQueryBitmapHandle 15-41
 GreQueryCharPositions 12-24
 GreQueryClipRegion 14-67
 GreQueryCodePageVector 15-54
 GreQueryColorData 12-94
 GreQueryColorIndex 12-102
 GreQueryDeviceBitmaps 12-105
 GreQueryDeviceCaps 12-106
 GreQueryDevResource 12-108
 GreQueryEngineVersion 15-15
 GreQueryFontAttributes 15-53
 GreQueryFontFileDescriptions 15-55
 GreQueryFonts 15-52
 GreQueryHardcopyCaps 12-113
 GreQueryLogColorTable 12-95
 GreQueryLogicalFont 15-46
 GreQueryNearestColor 12-101

query functions *(continued)*
GreQueryNumberSetIds 15-38
GreQueryRealColors 12-100
GreQueryRGBColor 12-103
GreQuerySetIds 15-39
GreQueryTextBox 12-22
GreQueryWidthTable 12-26
PostDeviceModes 11-2
QueryDeviceNames 11-4
QueryEscSupport escape code 12-127
QueryVioCellSizes escape code 12-128
SplQpQueryDt 17-27
WinQueryProcessCp 12-85
Queue Driver i
queue manager 17-11
SplQmAbort 17-12
SplQmAbortDoc 17-13
SplQmClose 17-14
SplQmEndDoc 17-15
SplQmOpen 17-16
SplQmStartDoc 17-17
SplQmWrite 17-18
queue processor 17-20
OS2SYS.INI B-1
SplQpClose 17-21
SplQpControl 17-22
SplQpInstall 17-23
SplQpOpen 17-24
SplQpPrint 17-26
SplQpQueryDt 17-27
queued devices 17-1

R

raster devices 10-10
raster operation 12-35
RawData escape code 12-129
RealizeFont 12-72
RectInRegion 14-51
RECTL structure 13-17, 14-46
RectVisible 14-66
reference phrase
definition of X-8
reference phrase help
definition of X-8
region functions 14-45
GreCombineRectRegion 14-56
GreCombineRegion 14-55
GreCombineShortLineRegion 14-57
GreCopyClipRegion 14-76
GreCreateRectRegion 14-52
GreDestroyRegion 14-53
GreEqualRegion 14-58
GreExcludeClipRectangle 14-70
GreGetClipBox 14-61
GreGetClipRects 14-62
GreGetRegionBox 14-46
GreGetRegionRects 14-47
GreIntersectClipRectangle 14-69
GreNotifyClipChange 12-69
GreOffsetClipRegion 14-64
GreOffsetRegion 14-49
GrePaintRegion 14-59
GrePtInRegion 14-50
GreQueryClipRegion 14-67
GreRectInRegion 14-51
GreRectVisible 14-66
region functions *(continued)*
GreRegionSelectBitmap 14-75
GreRestoreRegion 14-73
GreSaveRegion 14-72
GreSelectClipRegion 14-68
GreSelectPathRegion 14-74
GreSetRectRegion 14-54
VisRegionCallBack 16-14
RegionSelectBitmap 14-75
ResetBounds 12-82
ResetDC 15-7
ResetDCState 11-17
RestoreDC 15-14
RestoreDCState 11-16
RestorePath 14-44
RestoreRegion 14-73
RestoreScreenBits 13-12
RestoreXform 14-101
RestoreXformData 14-92
Resurrection 13-20
return code format 10-7
revalidating the visible region 16-14
RGNRECT structure 14-47
ring-3 16-13

S

sample presentation driver 10-7
SaveDC 15-13
SaveDCState 11-15
SavePath 14-43
SaveRegion 14-72
SaveScreenBits 13-11
SaveXform 14-100
SaveXformData 14-91
scan functions
GrePolyScanLine 12-14
SCANDATA structure 12-14
screen coordinates 10-7
screen switching 13-19, 13-20
scrolling
using GreBitblt 13-7
ScrollRect 13-7
segment server
SSAllocHuge 16-10
SSAllocSeg 16-11
SSFreeSeg 16-12
select font 15-45
SelectBitmap 12-34, 15-20
SelectClipPath 14-41
SelectClipRegion 14-68
SelectPathRegion 14-74
semaphores
FSRSemCheck 16-2
FSRSemEnter 16-3
FSRSemExit 16-5
FSRSemLeave 16-6
set ID functions 15-36
GreDeleteSetId 15-37
GreQueryNumberSetIds 15-38
GreQuerySetIds 15-39
SetArcParameters 14-4
SetAttributes 15-25
SetAVIOFont 13-9
SetBitmapBits 12-48
SetBitmapDimension 15-23
SetBitmapID 15-42

SetCodePage 12-86
SetColorCursor 13-16
SetCurrentPosition 12-6
SetDCOrigin 13-22
SetDefaultArcParameters 15-32
SetDefaultAttributes 15-29
SetDefaultLimits 15-34
SetDriverInfo 16-9
SetErrorInfo 16-15
SetGlobalAttribute 15-27
SetGlobalViewingXform 14-90
SetGraphicsField 14-96
SetHandle 15-10
SetLineOrigin 12-90
SetModelXform 14-86
SetPageUnits 14-83
SetPageViewport 14-94
SetPel 12-41
SetPickWindow 13-18
SetProcessControl 15-12
SetRectRegion 14-54
SetStyleRatio 12-78
SetupDC 14-77
SetViewingLimits 14-98
SetWindowViewportXform 14-88
SetXFormRect 14-71
severity of errors 10-13
SFACTORS structure 12-124
SHORTLINE structure 12-12, 14-57
SHORTLINEHEADER structure 12-12, 14-57
simulations
 bit map 10-10
 function 10-3
 presentation driver interface 14-1
SIZEL structure 14-82
source code, sample driver 10-7
spline 14-23
spline functions
 GrePolySpline 14-23
SplMessageBox 17-19
SplQmAbort 17-12
SplQmAbortDoc 17-13
SplQmClose 17-14
SplQmEndDoc 17-15
SplQmOpen 17-16
SplQmStartDoc 17-17
SplQmWrite 17-18
SplQpClose 17-21
SplQpControl 17-22
SplQpInstall 17-23
SplQpOpen 17-24
SplQpPrint 17-26
SplQpQueryDt 17-27
SplStdClose 17-4
SplStdDelete 17-5
SplStdGetBits 17-6
SplStdOpen 17-9
SplStdQueryLength 17-10
SplStdStart 17-7
SplStdStop 17-8
spooler 17-1
 components 17-1
 data types 17-1
 OS2SYS.INI B-1
 PM_Q_RAW data type 17-2
 PM_Q_STD data type 17-1
 SplMessageBox 17-19
spooler *(continued)*
 SplQmAbort 17-12
 SplQmAbortDoc 17-13
 SplQmClose 17-14
 SplQmEndDoc 17-15
 SplQmOpen 17-16
 SplQmStartDoc 17-17
 SplQmWrite 17-18
 SplQpClose 17-21
 SplQpControl 17-22
 SplQpInstall 17-23
 SplQpOpen 17-24
 SplQpPrint 17-26
 SplQpQueryDt 17-27
 SplStdClose 17-4
 SplStdDelete 17-5
 SplStdGetBits 17-6
 SplStdOpen 17-9
 SplStdQueryLength 17-10
 SplStdStart 17-7
 SplStdStop 17-8
 spool file 17-1
 spool file creation 17-1
SSAllocHuge 16-10
SSAllocSeg 16-11
SSFreeSeg 16-12
stack 10-5
 frame 10-5
StartDoc escape code 12-130
StartJournalFile 15-60
StdJournal escape code 12-131
STI 10-8
StopJournalFile 15-62
string functions
 GreCharRect 13-5
 GreCharStr 13-6
 GreCharString 12-17
 GreCharStringPos 12-19
 GreQueryCharPositions 12-24
 GreQueryTextBox 12-22
 GreQueryWidthTable 12-26
 GreScrollRect 13-7
 GreUpdateCursor 13-8
StrokePath 14-39
structure
 ARCPARAMS 14-4
 AREABUNDLE 12-54
 AREADEFS 12-54
 BANDRECT 12-126
 BITBLTATTRS 12-37
 BITMAPINFO 12-46
 CHARBUNDLE 12-56
 CHARDEFS 12-56
 DAREABUNDLE 12-54
 DCHARBUNDLE 12-56
 DELETERETURN 12-33
 DEVOPENSTRUC 11-9
 DIMAGEBUNDLE 12-60
 DISPLAYINFO 12-108
 DLINEBUNDLE 12-52
 DMARKERBUNDLE 12-61
 DRIVDATA 11-2
 ERRORID 16-15
 FATTRS 15-47
 FSRSEM 16-3
 HCINFO 12-113
 IMAGEBUNDLE 12-60

structure *(continued)*
LINE 12-10
LINEBUNDLE 12-52
LINEDEFS 12-52
MARKERBUNDLE 12-61
MARKERDEFS 12-61
NOTIFYTRANSFORMDATA 12-71
POINTL 14-5
POINTS 13-15
RECTL 13-17, 14-46
RGNRECT 14-47
SCANDATA 12-14
SFACTORS 12-124
SHORTLINE 12-12, 14-57
SHORTLINEHEADER 12-12, 14-57
SIZEL 14-82
style ratio 12-78
GreGetStyleRatio 12-77
styling
curves 12-2
lines 12-2
syntax conventions A-1
system functions 16-1

T

tag, definition of X-9
text functions 12-16
GreCharString 12-17
GreCharStringPos 12-19
GreQueryCharPositions 12-24
GreQueryTextBox 12-22
text mode 12-120
transform functions 14-78
GreConvert 14-99
GreConvertWithMatrix 14-102, 14-103
GreGetDCOrigin 13-21
GreGetDefaultViewingLimits 15-33
GreGetGlobalViewingXform 14-89
GreGetGraphicsField 14-95
GreGetModelXform 14-85
GreGetPageUnits 14-82
GreGetPageViewport 14-93
GreGetViewingLimits 14-97
GreGetWindowViewportXform 14-87
GreMultiplyXforms 14-100
GreNotifyTransformChange 12-70
GreRestoreXformData 14-92
GreSaveXformData 14-91
GreSetDefaultViewingLimits 15-34
GreSetGlobalViewingXform 14-90
GreSetGraphicsField 14-96
GreSetModelXform 14-86
GreSetPageUnits 14-83
GreSetPageViewport 14-94
GreSetViewingLimits 14-98
GreSetWindowViewportXform 14-88
GreSetXFormRect 14-71
transform structure 12-71

U

UnLoadFont 15-51
UnlockDevice 12-88
UpdateCursor 13-8
user interface
SplMessageBox 17-19
USER_BOUNDS 10-6

V

validating the visible region 16-14
version number 11-2, 12-106
VIO functions 13-2
VioGetPSAddress 16-13
visible region
dirty vis region 10-8
GreDeviceInvalidateVisRegion 13-14
GrePtVisible 14-65
VisRegionCallBack 16-14
visual interface
SplMessageBox 17-19

W

window
definition of X-10
WinMessageBox 17-19
WinQueryProcessCP 11-12, 12-85
WinSetErrorInfo 16-15

IBM United Kingdom
International Products Limited
PO Box 41, North Harbour
Portsmouth, PO6 3AU
England

Printed in Denmark
by Roussell

